# Theoretical Analysis of Hierarchical Language Recognition and Generation by Transformers without Positional Encoding

**Daichi Hayakawa** **Issei Sato**
The University of Tokyo
{hayakawadaichi001, sato}@g.ecc.u-tokyo.ac.jp

## Abstract

In this study, we provide constructive proof that Transformers can recognize and generate hierarchical language efficiently with respect to model size, even without the need for a specific positional encoding. Specifically, we show that causal masking and a starting token enable Transformers to compute positional information and depth within hierarchical structures. We demonstrate that Transformers without positional encoding can generate hierarchical languages. Furthermore, we suggest that explicit positional encoding might have a detrimental effect on generalization with respect to sequence length.

## 1 Introduction

Transformer-based models have achieved significant success in natural language processing. The empirical success of Transformers has drawn attention to the theoretical understanding of the problem classes that Transformers can solve. In particular, the ability to understand the hierarchical structures inherent in natural and programming languages is essential for accurately grasping their meanings. The empirical success of large language models (LLMs) and the positive findings (Murty et al., 2023), which empirically showed that the accuracy on the tasks possessing hierarchical structures improves through grokking, suggest that Transformers may have the ability to understand hierarchical structures. However, Tran et al. (2018), Petty and Frank (2021), and Mueller et al. (2022) stated that Transformers have different inductive biases than humans and often face difficulties processing such hierarchical languages, raising critical concerns about whether Transformers possess the capability to accurately capture hierarchical structures.

To investigate the expressive capacity of Transformers, several studies (Hahn, 2020, Bhattamishra et al., 2020, Ebrahimi et al., 2020, Yao et al., 2021, Chiang and Cholak, 2022, and Wen et al., 2023) have formulated such challenges as recognition/generation tasks of formal languages such as the parity language, the $\texttt{Dyck}_k$ language, and the $\texttt{Shuffle-Dyck}_k$ language. The parity language is a language over an alphabet consisting of only two characters, 0 and 1. A string belongs to the parity language if and only if it has an odd number of 1s. While the parity language is quite simple, it has a Kleene closure that characterizes regular languages. In contrast, the $\texttt{Dyck}_k$ language is a language over an alphabet consisting of $k$ types of brackets. Intuitively, it includes properly balanced strings. The $\texttt{Shuffle-Dyck}_k$ language is a shuffle of multiple $\texttt{Dyck}_1$ languages defined over different bracket-pairs. For example, regarding $\texttt{Dyck}_2$ and $\texttt{Shuffle-Dyck}_2$ over {"(", ")", "[", "]"}, "[]()" and "([()])" belong to $\texttt{Dyck}_2$, while "[(])" and "([(])" belong to $\texttt{Shuffle-Dyck}_2$ not to $\texttt{Dyck}_2$. Despite their simplicity, these languages are important because they provide a simplified framework for investigating the ability to comprehend hierarchical structures, which are found in both natural and programming languages, as well as the capability to process these structures in parallel.

Hahn (2020) pointed out that Lipschitz-bounded Transformers cannot solve recognition and generation tasks of Dyck languages for arbitrary lengths, implying that Transformers do not have the ability to grasp hierarchical structures. However, Yao et al. (2021) provided a proof that Transformer with specific absolute positional encoding can generate $\texttt{Dyck}_k$ and recognize $\texttt{Dyck}_{k,D}$, where $\texttt{Dyck}_{k,D}$ is a subset of $\texttt{Dyck}_k$ but the maximum nesting depth is bounded to $D$. Furthermore, Wen et al. (2023) provided an existence proof that a 2-layer $O(k^2D^2)$-width Transformer can process $\texttt{Dyck}_{k,D}$. These theoretical results raise the question:

> Why can Transformers with smaller widths and without specific absolute posi-



*Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30777–30834

tional encoding experimentally perform well on processing the $\texttt{Dyck}_k$ language?

In contrast to the approaches of Bhattamishra et al. (2020), Yao et al. (2021), and Wen et al. (2023), our theoretical analysis offers two advantages: (i) it reduces the linear or super-linear dependency of the number of bracket types $k$ and the maximum depth $D$ on the network width, and (ii) it does not rely on specific absolute positional encoding. Table 1 outlines these differences and highlights the strengths of our approach in comparison.

Our contributions are summarized as follows.

1. We provide constructive proofs that with a starting token, causal Transformers with a constant number of layers and $O(\log k)$ width have the ability to recognize the $\texttt{Dyck}_k$ and $\texttt{Shuffle-Dyck}_k$ languages and to generate the $\texttt{Dyck}_k$ language. Moreover, we also present a proof that those with a constant number of layers and $O(k)$ width have the ability to generate the $\texttt{Shuffle-Dyck}_k$ language. Note that the network is followed by a fully-connected layer whose output dimension is $\mathbb{R}$ for a recognition task and $K$ for a generation task, where $\mathbb{R}^K$ is the vocabulary size.

2. We give a constructive proof that Transformers still have the ability to create a signal that can serve similarly to a starting token by only leveraging causal masking under an additional assumption.

Our proofs include detailed descriptions of matrix notations and transformations at each layer for clarity, which makes our paper relatively lengthy. Therefore, we provide high-level proof sketches to allow readers to grasp the main ideas without delving into the details.

## 2 Related Work

Since the emergence of Transformer (Vaswani et al., 2017), a wide range of theoretical analyses have been conducted on its expressive capacity. Some of these analyses have focused on formal language recognition and generation tasks, particularly for the $\texttt{Dyck}_k$ language and the $\texttt{Shuffle-Dyck}_k$ language.

Bhattamishra et al. (2020) theoretically showed that a Transformer with a width of $O(k)$ can recognize the $\texttt{Shuffle-Dyck}_k$ language. In addition, Yao et al. (2021) provided a constructive proof that by using specific absolute positional encoding $i/n$, where $n$ is the maximum length of the input string, and $i$ is the position of characters, a $(D+1)$-layer causal Transformer can recognize the $\texttt{Dyck}_{k,D}$ language. Yao et al. (2021) also proved that using absolute positional encoding $i/n$, $i/n^3$, and $i$, a 2-layer causal Transformer can generate the $\texttt{Dyck}_k$ language. Furthermore, Wen et al. (2023) proved that a 2-layer Transformer network with a width of $O(k^2D^2)$ can generate $\texttt{Dyck}_{k,D}$.

## 3 Preliminaries

### 3.1 Dyck Languages

The $\texttt{Dyck}_k$ language is a context-free language over an alphabet consisting solely of $k$ types of bracket pairs $\{\langle_t, \rangle_t\}_{t=1}^k$ and includes strings with correctly nested brackets.

Despite its simplicity, Chomsky and Schützenberger (1959) showed that any context-free language can be expressed as a homomorphism of the intersection of the Dyck language and a regular language, suggesting that the Dyck language has an essence of context-free languages; that is, the Dyck langauge abstracts the hierarchical structures that context-free languages have but regular languages do not. Therefore, we aim to analyze the recognition and generation capacity of Transformers with respect to $\texttt{Dyck}_k$ and its variant, $\texttt{Shuffle-Dyck}_k$.

In this paper, we consider languages with two special tokens, <bos> and <eos>, which stand for "beginning-of-sentence" and "end-of-sentence" respectively. In language models, <bos> is typically inserted at the start, and <eos> is used as a signal to stop generating output. Therefore, we define the $\texttt{Dyck}_k$ and $\texttt{Shuffle-Dyck}_k$ languages for language models as follows:

**Definition 1** ($\texttt{Dyck}_k$ language for language models)**.** *The $\texttt{Dyck}_k$ language for language models is a context-free language over an alphabet $\Sigma = \{\langle_t, \rangle_t\}_{t=1}^k \cup \{\texttt{<bos>}, \texttt{<eos>}\}$. The following context-free grammar generates $\texttt{Dyck}_k$ language:*

$$S \to \texttt{<bos>}\, X\, \texttt{<eos>}, \quad (1)$$

$$X \to \varepsilon \mid \langle_1\, X\, \rangle_1\, X \mid \cdots \mid \langle_k\, X\, \rangle_k\, X, \quad (2)$$

*where $S$ and $\varepsilon$ are the starting symbol and empty string, respectively.*

**Definition 2** ($\texttt{Shuffle-Dyck}_k$ language for language models (informal))**.** *The $\texttt{Shuffle-Dyck}_k$*

| Method | Language | Width | Positional Encoding |
|---|---|---|---|
| **Recognition task** | | | |
| Bhattamishra et al. (2020) | $\texttt{Shuffle-Dyck}_k$ | $O(k)$ | None |
| Yao et al. (2021) | $\texttt{Dyck}_{k,D}$ | $O(\log k)$ | $i/n$ |
| Ours | $\texttt{Dyck}_k$ | $O(\log k)$ | None |
| | $\texttt{Shuffle-Dyck}_k$ | $O(\log k)$ | None |
| **Generation task** | | | |
| Yao et al. (2021) | $\texttt{Dyck}_{k,D}$ | $O(\log k)$ | $i/n$ |
| | $\texttt{Dyck}_k$ | $O(\log k)$ | $i/n, i/n^3, n$ |
| Wen et al. (2023) | $\texttt{Dyck}_{k,D}$ | $O(k^2D^2)$ | None |
| Ours | $\texttt{Dyck}_k$ | $O(\log k)$ | None |
| | $\texttt{Shuffle-Dyck}_k$ | $O(k)$ | None |

Table 1: Comparison of proposed method to previous studies. Note that **Width** represents the width of the Transformer block, excluding the width of the heads for each task. Since a task-specific head is a mapping from $\mathbb{R}^{d_{\text{model}}}$ to $\mathbb{R}$ for a recognition task and to $\mathbb{R}^K$ for a generation task, each has an $\Omega(d_{\text{model}})$-width and an $\Omega(\max(d_{\text{model}}, K))$-width head, respectively. In all cases in this table, the width of a task-specific head is $O(d_{\text{model}})$ for a recognition task and $O(\max(d_{\text{model}}, K))$ for a generation task.

*language for language models is defined as follows:*

$$\{\texttt{<bos>}w\texttt{<eos>} | w \in \texttt{Shuffle-Dyck}_k\}, \quad (3)$$

*where* $\texttt{Shuffle-Dyck}_k$ *is a language over an alphabet* $\Sigma = \{\langle_t, \rangle_t\}_{t=1}^k$ *and is defined as a shuffle of* $k$ *multiple* $\texttt{Dyck}_1$ — $\texttt{Dyck}_1^1, \cdots, \texttt{Dyck}_1^k$ —*, where* $\texttt{Dyck}_1^t$ *is the* $\texttt{Dyck}_1$ *language over an alphabet* $\{\langle_t, \rangle_t\}$*. A formal definition of* $\texttt{Shuffle-Dyck}_k$ *is provided in Appendix B.1.*

For example, "$\langle_1 \langle_2 \rangle_1 \rangle_2$" does not belong to $\texttt{Dyck}_2$ but to $\texttt{Shuffle-Dyck}_2$. $\texttt{Shuffle-Dyck}_k$ can be recognized by $k$-counter machines; thus, this language provides insights into the ability to process $k$ hierarchical structures in parallel.

We also define a prefix for languages and the depth of a prefix in the $\texttt{Dyck}_k$ language as follows:

**Definition 3** (Prefix for language)**.** *A string* $w \in \Sigma^*$ *is a prefix for language* $\mathcal{L}$ *if there exists* $u \in \Sigma^*$ *such that* $wu \in \mathcal{L}$*. In addition, denote by* $w \in \mathrm{Pre}(\mathcal{L})$ *that* $w$ *is a prefix for* $\mathcal{L}$*.*

Hereafter, denote an input string of length $n+1$ by $w_{0:n}$ and the prefix of length $i+1$ by $w_{0:i}$. Note that strings starting at index 0 implicitly include <bos> at the beginning; that is, for instance, $w_{0:i}$ represents <bos>$w_{1:i}$.

**Definition 4** (Depth of string)**.** *The depth of a prefix* $w_{0:i} (=$ <bos>$w_{1:i})$ *in* $\texttt{Dyck}_k$ *is defined as follows:*

$$\mathrm{d}(w_{0:i}) = \#_{\langle}(w_{0:i}) - \#_{\rangle}(w_{0:i}), \quad (4)$$

*where* $\#_{\langle}(w_{0:i})$ *and* $\#_{\rangle}(w_{0:i})$ *represent the number of open brackets and closed brackets in* $w_{0:i}$*, respectively. Here, the differences in bracket types are ignored.*

Note that for any prefix $w_{0:i}$ for $\texttt{Dyck}_k$, the following three statements hold: (i) $w_{0:i}\langle$ is always a prefix, (ii) if $\mathrm{d}(w_{0:i}) = 0$, $w_{0:i}\rangle$ cannot be a prefix, and (iii) if $\mathrm{d}(w_{0:i}) \geq 1$, there exists only one type $t_{\text{valid}}$ such that $w_{0:i}\rangle_{t_{\text{valid}}}$ is a prefix. With respect to (iii), although such a closed bracket depends on $w_{0:i}$, denote it by $\rangle_{t_{\text{valid}}}$ in an abusive manner. In addition, there can be more than one $\rangle_{t_{\text{valid}}}$ in $\texttt{Shuffle-Dyck}_k$.

### 3.2 Transformer architecture

Let $d_{\text{model}}$ be the dimension of the embedding vectors and hidden representations, $\Sigma$ be the vocabulary set. Transformer architecture takes an input string of length $n$ and converts each character into a $d_{\text{model}}$-dimensional vector. Then, by applying Transformer blocks $\left(\mathbb{R}^{n \times d_{\text{model}}} \to \mathbb{R}^{n \times d_{\text{model}}}\right)$ for multiple times, an output of dimension $\mathbb{R}^{n \times d_{\text{model}}}$ is obtained. Since $n$ is not fixed, we represent a Transformer as $\mathcal{T} : \Sigma^* \to \mathbb{R}^{* \times d_{\text{model}}}$.

In this paper, we largely follow the Transformer architecture adopted in Yao et al. (2021); namely, we consider a Transformer architecture composed of multiple single-head Transformer blocks, each of which incorporates a self-attention layer and a feed-forward network layer. The two major differences of the architecture adopted in Yao et al. (2021) from the model proposed by Vaswani et al. (2017) are as follows: (i) using single-head attention instead of multi-head attention and (ii) incorporating the layer normalization (Ba et al., 2016)

right after the first linear transformation in the feed-forward network layer instead of after the attention layer and feed-forward network layer. The latter assumption is also incorporated in Merrill and Sabharwal (2024).

We adopt the architecture in Yao et al. (2021) with a slight modification: we replace the standard layer normalization (Ba et al., 2016) with the RMS layer normalization (Zhang and Sennrich, 2019). Zhang and Sennrich (2019) empirically showed that the RMS layer normalization reduces the training time compared to the conventional layer normalization while maintaining the same performance. The RMS layer normalization has been adopted in recent models such as Llama (Touvron et al., 2023a) and Llama 2 (Touvron et al., 2023b). The details of the Transformer architecture are provided in Appendix B.2.

### 3.3 Language recognition and generation

In this paper, we mainly focus on two tasks: language recognition and generation. Here, we define language recognition and generation by Transformers. For each task, a fully-connected layer follows the network, and the output dimension is $\mathbb{R}$ for recognition tasks and $\mathbb{R}^K$ for generation tasks, which we call the recognizer head and generator head, respectively.

**Definition 5** (Language recognition by Transformers)**.** *A Transformer $\mathcal{T} : \Sigma^* \to \mathbb{R}^{*\times d_{\text{model}}}$ recognizes a language $\mathcal{L} \subseteq \Sigma^*$ if there exists a fully-connected layer $f_{\text{rec}} : \mathbb{R}^{d_{\text{model}}} \to \mathbb{R}$ such that for any $w_{0:n} \in \Sigma^*$,*

$$\operatorname{sgn}(f_{\text{rec}}(\mathcal{T}(w_{0:n})_n)) = \begin{cases} 1 & \text{if } w_{0:n} \in \mathcal{L} \\ -1 & \text{if } w_{0:n} \notin \mathcal{L} \end{cases}, \tag{5}$$

*where $\operatorname{sgn}(\cdot)$ is a sign function.*

It is impossible to define language generation by Transformers by simply setting a threshold on the output probability of each string in a language because formal languages are typically infinite string sets. Therefore, we first define language generation process and then define language generation by Transformers. This approach is similar to the methods in Yao et al. (2021), Wen et al. (2023) and Svete and Cotterell (2024). Specifically, we define language generation process using the conditional categorical distribution as follows.

**Definition 6** (Language generation process)**.** *A language generation process over an alphabet $\Sigma$ is a categorical distribution over $\Sigma$ conditioned by a string $w_{0:i}$. Specifically, denote the language generation process of a language $\mathcal{L}$ by $p_{\mathcal{L}}(w_{i+1} \mid w_{0:i})$.*

Note that language generation processes are well-defined: the following proposition holds.

**Proposition 1.** *For any language $\mathcal{L} \subseteq \Sigma^*$ over a finite alphabet $\Sigma$ and any probability distribution $p$ over $\mathcal{L}$, there exists a language generation process that produces the given probability distribution $p$. In other words, there exists a language generation process $p_{\mathcal{L}}(w_{i+1} \mid \texttt{<bos>}w_{1:i})$ such that for any string $w_{1:n} \in \mathcal{L}$,*

$$p(w_{1:n}) = p_{\mathcal{L}}(\texttt{<bos>}w_{1:n}\texttt{<eos>}), \tag{6}$$

*where*

$$\begin{aligned} &p_{\mathcal{L}}(\texttt{<bos>}w_{1:n}\texttt{<eos>}) \\ &\quad = p_{\mathcal{L}}(\texttt{<bos>}) \\ &\qquad \cdot \left( \prod_{i=1}^{n} p_{\mathcal{L}}(w_i \mid \texttt{<bos>}w_{1:i-1}) \right) \\ &\qquad \cdot p_{\mathcal{L}}(\texttt{<eos>} \mid \texttt{<bos>}w_{1:n}). \end{aligned} \tag{7}$$

*Proof.* The proof is provided in Appendix E. □

Then, we define the language generation by Transformers. We largely follow the definition in Yao et al. (2021), which defines it as whether the probability $p(w_i \mid w_{1:i-1})$ exceeds a certain threshold for any string $w_{1:n} \in \mathcal{L}$ and $i \in [n] (= \{1, \cdots, n\})$. However, we make this definition more stringent: we assume the existence of a true distribution and define it as the ability to output this distribution. This is because one of the most important properties of language models is the ability to generate diverse but natural sentences by assigning appropriate probability to consistent sequences. This approach is similar to Wen et al. (2023) and Svete and Cotterell (2024).

In general, Transformer-based language models transform the last token output with a fully-connected layer $f_{\text{gen}} : \mathbb{R}^{d_{\text{model}}} \to \mathbb{R}^K$. Then, the vector is converted into a probability vector with softmax function : $\mathbb{R}^K \to \Delta^{K-1}$, where $\Delta^{K-1} \left(\subset \mathbb{R}^K\right)$ is a probability simplex. Here, the softmax function transforms each element into a value in the range of $(0, 1)$, which makes it impossible to represent a probability of 0 or 1 exactly. Therefore, we define the realization of the language generation process by Transformers as the ability to approximate the language generation process with arbitrary precision as follows.

**Definition 7** (Realization of language generation process by Transformers)**.** *A Transformer* $\mathcal{T} : \Sigma^* \to \mathbb{R}^{*\times d_{\text{model}}}$ *realizes a language generation process* $p_{\mathcal{L}}(w_{i+1} \mid w_{0:i})$ *if for any* $\epsilon > 0$ *there exists a fully-connected layer* $f_{\text{gen}} : \mathbb{R}^{d_{\text{model}}} \to \mathbb{R}^K$ *such that if* $p_{\mathcal{L}}(w_{0:i}) > 0$ *then*

$$\text{TV}\left(p_{\mathcal{T}}(w_{i+1} \mid w_{0:i}), p_{\mathcal{L}}(w_{i+1} \mid w_{0:i})\right) < \epsilon, \quad (8)$$

*where* $p_{\mathcal{T}}(w_{i+1} \mid w_{0:i})$ *is the categorical distribution based on the output of Transformer and* $\text{TV}(\cdot, \cdot)$ *is the total variation distance. Specifically,*

$$p_{\mathcal{T}}(w_{i+1} \mid w_{0:i}) = \mathbb{S}(f_{\text{gen}}(\mathcal{T}(w_{0:i})_i)), \quad (9)$$

*where* $\mathbb{S}(\cdot)$ *is a softmax function and the total variation distance between two* $K$*-dimensional categorical distributions* $p = (p_1, \cdots, p_K)$ *and* $p' = (p'_1, \cdots, p'_K)$ *is expressed as follows:*

$$\text{TV}(p, p') = \frac{1}{2} \sum_{l=1}^{K} \left| p_l - p'_l \right| \quad (10)$$

Next, we define the language generation process of $\texttt{Dyck}_k$. Note that this definition generalizes the definition in Hewitt et al. (2020) and Wen et al. (2023): they treat all types of brackets in a symmetric way, while we slightly generalize the approach to be able to assign different probabilities.

**Definition 8** ($\texttt{Dyck}_k$ language generation process)**.** *A language generation process* $p(w_{i+1} \mid w_{0:i})$ *over an alphabet* $\Sigma = \{\langle_t, \rangle_t\}_{t=1}^k \cup \{\texttt{<bos>}, \texttt{<eos>}\}$ *is called the* $\texttt{Dyck}_k$ *language generation process if*

$$p(w_0 = \text{“}\texttt{<bos>}\text{”} \mid \varepsilon) = 1, \quad (11)$$

$$p(w_{i+1} \mid w_{0:i}) = \begin{cases} p_0(w_{i+1}) & \text{if } \mathrm{d}(w_{0:i}) = 0 \\ p_1(w_{i+1}) & \text{if } \mathrm{d}(w_{0:i}) \geq 1 \end{cases}, \quad (12)$$

*where*

$$p_0(w_{i+1}) = \begin{cases} r\pi_t & \text{if } w_{i+1} = \text{“}\langle_t\text{”} \\ 1 - r & \text{if } w_{i+1} = \text{“}\texttt{<eos>}\text{”} \\ 0 & \text{otherwise} \end{cases},$$

$$p_1(w_{i+1}) = \begin{cases} q\pi_t & \text{if } w_{i+1} = \text{“}\langle_t\text{”} \\ 1 - q & \text{if } w_{i+1} = \text{“}\rangle_{t_{\text{valid}}}\text{”} \\ 0 & \text{otherwise} \end{cases}, \quad (13)$$

$q, r \in (0, 1)$, $\boldsymbol{\pi} \in \Delta^{k-1}$.

*Hereafter, we explicitly write the Dyck language generation process parameterized by* $q, r, \boldsymbol{\pi}$ *as* $p_{\texttt{Dyck}_k}(\cdot; q, r, \boldsymbol{\pi})$.

Note that the $\texttt{Dyck}_k$ language generation process defined above corresponds appropriately with the $\texttt{Dyck}_k$ language as described below.

**Proposition 2.** *For any length* $n$ *and* $\texttt{Dyck}_k$ *language generation process* $p_{\texttt{Dyck}_k}(\cdot; q, r, \boldsymbol{\pi})$*, there exists* $\epsilon_n$ *such that if* $\boldsymbol{\pi} > 0$ *then*

$$p_{\texttt{Dyck}_k}(\texttt{<bos>}w_{1:n}\texttt{<eos>}; q, r, \boldsymbol{\pi}) \begin{cases} \geq \epsilon_n & \text{if } w_{1:n} \in \texttt{Dyck}_k \\ = 0 & \text{if } w_{1:n} \notin \texttt{Dyck}_k \end{cases} \quad (14)$$

*holds.*

*Proof.* The Proof is provided in Appendix F. □

We also define the language generation process of $\texttt{Shuffle-Dyck}_k$ in a similar way. The details are provided in Appendix B.1.

## 4 Theoretical Results

In this section, we show our theoretical results.

**Theorem 1** (Transformers with starting token, $\texttt{Dyck}_k$ recognition)**.** *For all* $k$*, there exists a 5-layer* $O(\log k)$*-width causal Transformer without positional encoding that recognizes the* $\texttt{Dyck}_k$ *language. Each layer incorporates both the residual connection and the layer normalization. This network is followed by a fully-connected layer and a sign function to output an acceptance signal.*

*Proof sketch.* A Transformer network that recognizes the $\texttt{Dyck}_k$ language can be constructed by performing the following operations in each layer. Note that $w_{0:i}$ corresponds to $\texttt{<bos>}w_{1:i}$. First, we compute positional and depth information using <bos>. Then, using the information, we check whether the following two conditions are simultaneously satisfied: (i) $w_{1:i}$ is a prefix of the $\texttt{Dyck}_k$ language and (ii) the depth of $w_{0:i}$ is 0. Figure 1 provides the high-level understanding of this Transformer.

**First layer** creates pseudo positional encoding $\begin{bmatrix}\cos\phi(i) & \sin\phi(i)\end{bmatrix}^\top$ at position $i$, where $\phi(i) = \tan^{-1}(i/\exp(a))$ and $a$ is an attention score on <bos>.

**Second and third layers** count depth $\mathrm{d}(w_{0:i})$ and $\mathrm{d}(w_{0:i}) + 1$, respectively. This is because the depth of the closed bracket is smaller by 1 than the corresponding open bracket. For instance, the depths calculated for “$\langle_1\rangle_1$” are 1 for “$\langle_1$” and 0 for “$\rangle_1$”. These computations are achieved

by constructing a value matrix that outputs 1 for open brackets and $-1$ for closed brackets in a specific dimension.

**Fourth layer** calculates a value that corresponds to a propositional variable $Q(w_{0:i})$ that indicates $w_{1:i} \in \texttt{Dyck}_k$ but is guaranteed to return the expected value ($w_{1:i} \in \texttt{Dyck}_k$) only when $i = 0$ or $w_{1:i-1}$ is a prefix for $\texttt{Dyck}_k$. Therefore, to check whether $w_{1:i} \in \texttt{Dyck}_k$ in the subsequent layers, we have to check whether all propositional variables $\{Q(w_{0:j})\}_{j=0}^{i}$ return `True` or not.

**Fifth layer** calculates (i) whether $w_{1:n}$ is a prefix for $\texttt{Dyck}_k$ with $\bigwedge_{i=1}^{n} Q(w_{0:i})$ and (ii) whether $\mathrm{d}(w_{0:i}) = 0$ or not.

The subsequent fully-connected layer determines whether the string $w_{1:n}$ belongs to $\texttt{Dyck}_k$ by examining whether the two conditions calculated in the fifth layer are simultaneously satisfied.

The full proof is provided in Appendix G. □

**Theorem 2** (Transformers with starting token, $\texttt{Dyck}_k$ generation)**.** *For all $k$, there exists a 3-layer $O(\log k)$-width causal Transformer network without positional encoding that generates the $\texttt{Dyck}_k$ language. Each layer incorporates both the residual connection and the layer normalization. This network is followed by a fully-connected layer and softmax layer to output the probability distribution.*

*Proof sketch.* A Transformer network that generates the $\texttt{Dyck}_k$ language can be constructed by performing the following operations in each layer. The first and second layers do the same operations as those used in Theorem 1. Figure 2 provides the high-level understanding of this Transformer.

**First layer** creates pseudo positional encoding $\begin{bmatrix}\cos\phi(i) & \sin\phi(i)\end{bmatrix}^\top$.

**Second layer** counts depth $\mathrm{d}(w_{0:i})$.

**Third layer** fetches a valid closed bracket if one exists; otherwise, a zero vector is fetched. This operation is achieved by placing attention on the largest position among $\{0\} \cup \{j \mid \mathrm{d}(w_{0:j}) = \mathrm{d}(w_{0:i})\}$.

Then, the subsequent fully-connected layer and softmax operation output the next token distribution using the vector calculated in the third layer.

The full proof is provided in Appendix H. □

**Proposition 3** (Transformers with starting token, $\texttt{Shuffle-Dyck}_k$ recognition)**.** *For all $k$, there exists a 3-layer $O(\log k)$-width causal Transformer without positional encoding that recognizes the $\texttt{Shuffle-Dyck}_k$ language. Each layer incorporates both the residual connection and the layer normalization. This network is followed by a fully-connected layer and a sign function to output an acceptance signal.*

*Proof.* The proof is provided in Appendix I. □

**Proposition 4** (Transformers with starting token, $\texttt{Shuffle-Dyck}_k$ generation)**.** *For all $k$, there exists a 3-layer $O(k)$-width causal Transformer without positional encoding that generates the $\texttt{Shuffle-Dyck}_k$ language. Each layer incorporates both the residual connection and the layer normalization. This network is followed by a fully-connected layer and softmax layer to output the probability distribution.*

*Proof.* The proof is provided in Appendix J. □

**Proposition 5.** *There is no network whose width grows strictly slower than $k/\log k$ that generates $\texttt{Shuffle-Dyck}_k$; that is, if*

$$\lim_{k\to\infty} \frac{d_{\mathrm{model}}(k)}{k/\log k} = 0 \tag{15}$$

*holds, then there exists $k_0$ such that for any $k \geq k_0$, networks with $d_{\mathrm{model}}(k)$-width cannot generate $\texttt{Shuffle-Dyck}_k$.*

*Proof.* The proof is provided in Appendix K. □

Next, we show that even without <bos>, Transformers can recognize and generate $\texttt{Dyck}_k$ languages under certain conditions. The following proposition states that under relatively weak conditions, Transformers can generate a signal that serves a similar role to <bos> in Theorems 1, 2.

**Proposition 6.** *Assume that there exists a linear mapping such that the transformed embeddings are distinct from each other and have a constant 2-norm. Then, there exists a Transformer block without a starting token that creates a pseudo starting signal $\hat{s}_i$ for any string $w_{1:n}$ whose first two tokens are different, where*

$$\hat{s}_i = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{otherwise} \end{cases}. \tag{16}$$

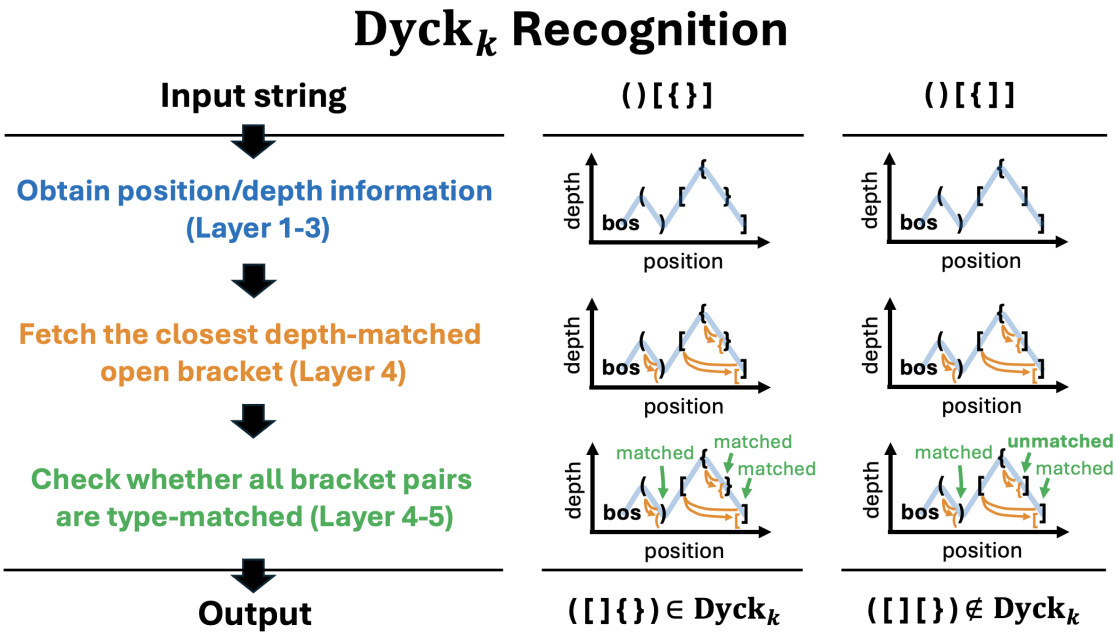


Figure 1: Intuitive illustration of the Transformer that recognizes $\texttt{Dyck}_k$. The specific implementation is described in Section G.

*Specifically, this block transforms the constants-padded vector* $\hat{\mathbf{x}}_i$ *as follows:*

$$\hat{\mathbf{x}}_i = \begin{bmatrix} \mathbf{x}_i \\ \vdots \\ 0 \end{bmatrix} \mapsto \begin{bmatrix} \mathbf{x}_i \\ \vdots \\ \hat{s}_i \end{bmatrix}. \tag{17}$$

*Proof.* The proof is provided in Appendix L. □

By leveraging Proposition 6, we also show that Transformers without <bos> can recognize and generate under the same assumption of Proposition 6.

**Corollary 1** (Transformers without starting token, $\texttt{Dyck}_k$ probabilistic recognition)**.** *Assume the same assumption as in Proposition 6. There exists a* 9*-layer causal Transformer without a starting token that recognizes the* $\texttt{Dyck}_k$ *language with probability at least* $1 - 1/k$.

*Proof.* The proof is provided in Appendix M. □

**Corollary 2** (Transformers without starting token, $\texttt{Dyck}_k$ subset generation)**.** *Assume the same assumption as in Proposition 6. There exists a* 7*-layer causal Transformer without a starting token that can generate a subset of* $\texttt{Dyck}_k$ *where the first two characters are different; that is, the Transformer can generate all possible subsequent sequences when there is an input string whose first two characters are different.*

*Proof.* The proof is provided in Appendix N. □

## 5 Experiments

The constructive proofs in the previous section show that even without the need for a specific positional encoding, single-head Transformers with a starting token have the ability to recognize and generate $\texttt{Dyck}_k$ and $\texttt{Shuffle-Dyck}_k$, and that even without a starting token, Transformers can recognize and generate $\texttt{Dyck}_k$. In this section, we examined the theoretical results by conducting experiments on the generation ability for $\texttt{Dyck}_k$ with/without a starting token (Theorem 2 and Corollary 2). We also investigated the generation ability on $\texttt{Shuffle-Dyck}_k$ (Proposition 4).

In addition, we empirically investigated the effect of the layer normalization position on model performance using natural language datasets because the Transformer architecture used in this paper differs from common architectures regarding the layer normalization position.

### 5.1 Evaluation on $\texttt{Dyck}_k$ and $\texttt{Shuffle-Dyck}_k$

Our constructive proofs show that Transformers are capable of recognizing and generating $\texttt{Dyck}_k$ and $\texttt{Shuffle-Dyck}_k$. In this section, we experimentally investigated whether such networks can actually be learned. Here, we provide a brief explanation of the experimental setup and the results

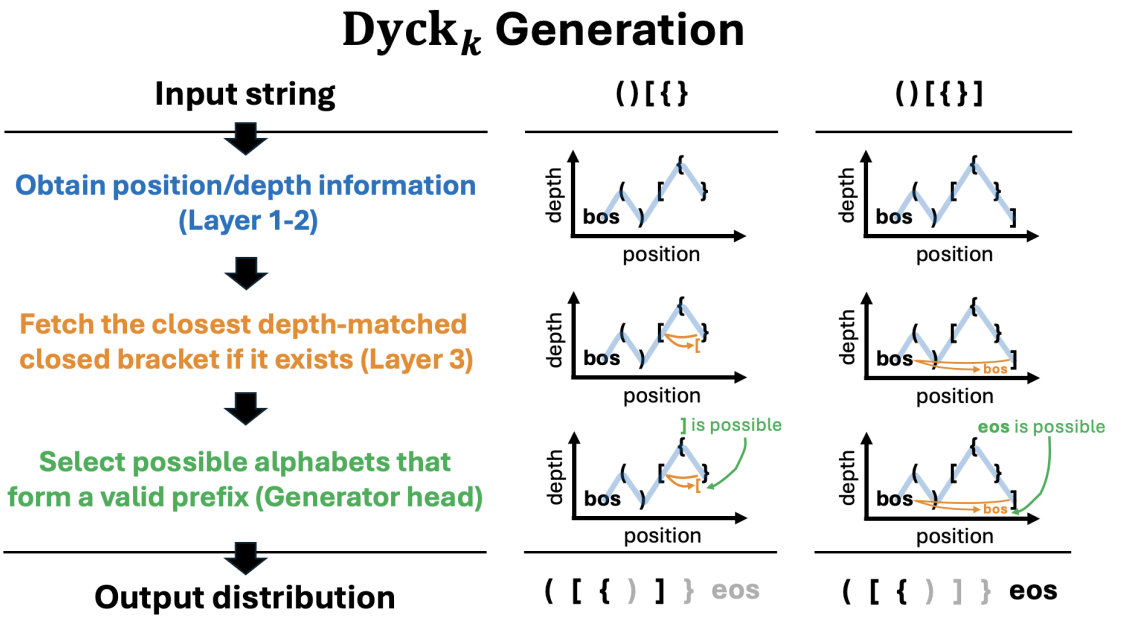


Figure 2: Intuitive illustration of the Transformer that generates $\texttt{Dyck}_k$. The specific implementation is described in Section H.

for $\texttt{Dyck}_8$ and $\texttt{Shuffle-Dyck}_8$, while the detailed explanation and other results are provided in Appendix R.

**Setup** Following Yao et al. (2021), we generated training and validation sets with the maximum input length of 700 according to a language generation process. We trained Transformers with causal masking by having them solve a next-token prediction task.

We compared four types of models: (i) with positional encoding and a starting token (PE+BOS), (ii) with positional encoding but without a starting token (PE+NoBOS), (iii) without positional encoding but with a starting token (NoPE+BOS), and (iv) without positional encoding and a starting token (NoPE+NoBOS). We reported the average accuracy of generating correct closed brackets, which is described below, separately for in-distribution (ID) data ($n \leq 700$) and out-of-distribution (OOD) data ($700 < n \leq 840$).

**Metric** Following Hewitt et al. (2020), Yao et al. (2021), we report the conditioned probability to output the correct closed bracket(s):

$$\mathrm{Acc}_{\mathrm{closed}} = \mathbb{E}\left[p(\rangle_{t_{\mathrm{valid}}} \mid \rangle \cdot)\right], \tag{18}$$

where

$$p(\rangle_{t_{\mathrm{valid}}} \mid \rangle \cdot) = \begin{cases} \dfrac{p(\rangle_{t_{\mathrm{valid}}})}{\sum_{t=1}^{k} p(\rangle_t)} & \text{for } \texttt{Dyck}_k \\ \dfrac{\sum_{t_{\mathrm{valid}}} p(\rangle_{t_{\mathrm{valid}}})}{\sum_{t=1}^{k} p(\rangle_t)} & \text{for } \texttt{Shuffle-Dyck}_k \end{cases}. \tag{19}$$

This metric indicates how accurately the models can generate the sequence.

Figure 3 shows the test accuracy of generating the correct closed bracket on $\texttt{Dyck}_8$ and $\texttt{Shuffle-Dyck}_8$, while the results for other values of $k$ are provided in Appendix R.

### 5.2 Evaluation on natural language datasets

In the previous section, we derived theoretical results using the architecture that differs from the common ones with respect to the position of the layer normalization. In this section, we investigated the performance differences that arise from the layer normalization positions because Wang et al. (2019) and Xiong et al. (2020) empirically showed that layer normalization position significantly affects the model performance.

Specifically, we investigated whether the architecture used in our proofs, which we call FFN-LN, could benefit from the layer normalization by comparing FFN-LN with the architectures with two common layer normalization positions, Post-LN

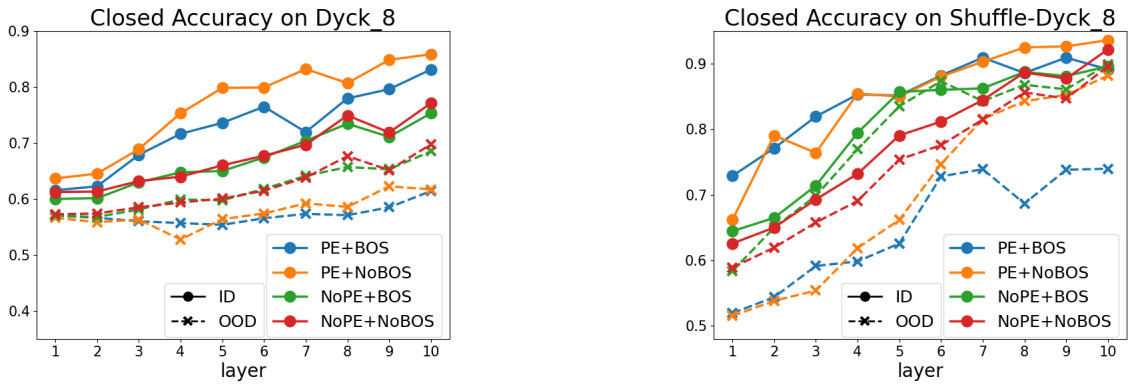


Figure 3: (Left) Test accuracy of generating the correct closed brackets on $\texttt{Dyck}_8$. (Right) Test accuracy of generating the correct closed bracket on $\texttt{Shuffle-Dyck}_8$. The solid lines represent the results for in-distribution data ($n \leq 700$), while the dashed lines represent the results for out-of-distribution data ($700 < n \leq 840$). In both experiments, results are averaged over 5 runs with different random seeds.

| Architecture | WikiText-103 | OpenWebText |
|---|---|---|
| Post-LN | 19.11 | 20.82 |
| Pre-LN | 19.44 | 20.83 |
| No-LN | 21.25 | 22.72 |
| FFN-LN | 19.17 | 21.32 |

Table 2: Test perplexity on two natural language datasets with different positions of the layer normalization.

and Pre-LN, and without the layer normalization, No-LN. In Post-LN, the layer normalization is applied after the attention layer and the feed-forward network layer, while in Pre-LN, the layer normalization is applied before these layers.

We trained four 124M models (Post-LN, Pre-LN, No-LN, and FFN-LN) once each from scratch on two natural language datasets, WikiText-103 (Merity et al., 2016) and OpenWebText (Gokaslan et al., 2019). The test perplexities of the models that achieve the best validation losses are described in Table 2. The Appendix R provides detailed information about the training process and other results.

## 6 Discussion

### 6.1 Experiments on $\texttt{Dyck}_8$ and $\texttt{Shuffle-Dyck}_8$

From the results in Figure 3, PE lets models achieve higher accuracy on ID data compared to NoPE. However, the performance with PE on OOD data drops significantly. On the other hand, for NoPE, the performance on OOD data drops slightly compared to that on ID data. This suggests that NoPE might let models obtain a better inductive bias with respect to capturing hierarchical structure and generalizing with respect to sequence length. In addition, we did not observe a noticeable difference between BOS and NoBOS. This correlates with Corollary 2.

### 6.2 Experiments on natural language datasets

Here, we discuss the optimal position of the layer normalization. Wang et al. (2019) and Xiong et al. (2020) showed that Pre-LN leads to stable training and training time reduction compared to Post-LN, while Nguyen and Salazar (2019) and Mao et al. (2023) demonstrated that under certain conditions, such as machine translation, Post-LN outperforms Pre-LN. Furthermore, Shleifer et al. (2021) demonstrated that incorporating the layer normalization before the second linear layer of the feed-forward network layer can effectively mitigate gradient explosion and vanishing, which are commonly observed issues in both Pre-LN and Post-LN setups.

In this way, although the optimal position remains unclear, we conclude that our modified architecture is competitive to Pre-LN and Post-LN because the architecture used in our proof effectively benefits from the layer normalization in the experiments on WikiText-103 and OpenWebText. Further discussion on the layer normalization position is provided in Appendix S.

## 7 Conclusion

In this study, we theoretically showed that Transformers can efficiently process hierarchical languages even without the need for a specific positional encoding. Our theoretical and empirical results might alleviate the existing concern that Transformers, unlike RNNs and LSTMs, often face difficulties in capturing hierarchical structures.

## Limitations

We adopt the layer normalization position that differs from the commonly used positions, but it remains unclear whether this specific position is essential for our proofs. We also assume real numbers with infinite precision, occasionally involving operations with large real values, which leads to a question as to whether it is possible to realize such operations with finite-bit floating point representation. This issue is particularly important in light of recent trends towards quantization for reducing model sizes, where 16-bit or even 4 or 8-bit floating-point representations are frequently used.

In addition, we trained 124M models using two natural language datasets and empirically demonstrated the validity of the architecture we adopted. However, it remains unclear whether the adopted architecture is competitive with `Pre-LN` and `Post-LN` when applied to larger models or different datasets.

## Ethics Statement

This paper consists solely of theoretical results and supporting experiments. While we conducted experiments using natural language datasets, we have presented only sufficiently aggregated results. To the best of our knowledge, there are no ethical concerns or potential risks associated with this study.

## Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 20H05703 Japan. We thank anonymous reviewers for insightful suggestions and comments. We used a generative AI tool for language refinement.

## References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *Preprint*, arXiv:1607.06450.

Armen Bagdasaryan. 2023. On the partition of space by hyperplanes. *European Journal of Pure and Applied Mathematics*, 16(2):893–898.

Pablo Barcelo, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir Podolskii. 2024. Logical languages accepted by transformer encoders with hard attention. In *The Twelfth International Conference on Learning Representations*.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

David Chiang and Peter Cholak. 2022. Overcoming a theoretical limitation of self-attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7654–7664, Dublin, Ireland. Association for Computational Linguistics.

N. Chomsky and M.P. Schützenberger. 1959. The algebraic theory of context-free languages**this work was supported in part by the u.s. army signal corps, the air force office of scientific research, and the office of naval research; and in part by the national science foundation; and in part by a grant from the commonwealth fund. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 26 of *Studies in Logic and the Foundations of Mathematics*, pages 118–161. Elsevier.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy. Association for Computational Linguistics.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme Ruiz, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd Van Steenkiste, Gamaleldin Fathy Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Collier, Alexey A. Gritsenko, Vighnesh Birodkar, Cristina Nader Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetic, Dustin Tran, Thomas Kipf, Mario Lucic, Xiaohua Zhai, Daniel Keysers, Jeremiah J. Harmsen, and Neil Houlsby. 2023. Scaling vision transformers to 22 billion parameters. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 7480–7512. PMLR.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Javid Ebrahimi, Dhruv Gelda, and Wei Zhang. 2020. How can self-attention networks recognize Dyck-n languages? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4301–4306, Online. Association for Computational Linguistics.

Daniel Y Fu, Tri Dao, Khaled Kamal Saab, Armin W Thomas, Atri Rudra, and Christopher Re. 2023. Hungry hungry hippos: Towards language modeling with state space models. In *The Eleventh International Conference on Learning Representations*.

Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. 2019. Openwebtext corpus.

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

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *Preprint*, arXiv:1512.03385.

John Hewitt, Michael Hahn, Surya Ganguli, Percy Liang, and Christopher D. Manning. 2020. RNNs can generate bounded hierarchical languages with optimal memory. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1978–2010, Online. Association for Computational Linguistics.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. 2023. The impact of positional encoding on length generalization in transformers. In *Advances in Neural Information Processing Systems*, volume 36, pages 24892–24928. Curran Associates, Inc.

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. 2019. Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4365–4374, Hong Kong, China. Association for Computational Linguistics.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. 2024. Chain of thought empowers transformers to solve inherently serial problems. In *The Twelfth International Conference on Learning Representations*.

Zhuoyuan Mao, Raj Dabre, Qianying Liu, Haiyue Song, Chenhui Chu, and Sadao Kurohashi. 2023. Exploring the impact of layer normalization for zero-shot neural machine translation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1300–1316, Toronto, Canada. Association for Computational Linguistics.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

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

William Merrill and Ashish Sabharwal. 2024. The expressive power of transformers with chain of thought. In *The Twelfth International Conference on Learning Representations*.

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

Aaron Mueller, Robert Frank, Tal Linzen, Luheng Wang, and Sebastian Schuster. 2022. Coloring the blank slate: Pre-training imparts a hierarchical inductive bias to sequence-to-sequence models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1352–1368, Dublin, Ireland. Association for Computational Linguistics.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher Manning. 2023. Grokking of hierarchical structure in vanilla transformers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 439–448, Toronto, Canada. Association for Computational Linguistics.

Toan Q. Nguyen and Julian Salazar. 2019. Transformers without tears: Improving the normalization of self-attention. In *Proceedings of the 16th International Conference on Spoken Language Translation*, Hong Kong. Association for Computational Linguistics.

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

Jackson Petty and Robert Frank. 2021. Transformers generalize linearly. *Preprint*, arXiv:2109.12036.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Herbert Robbins. 1955. A remark on stirling's formula. *The American Mathematical Monthly*, 62(1):26–29.

Sam Shleifer, Jason Weston, and Myle Ott. 2021. Normformer: Improved transformer pretraining with extra normalization. *Preprint*, arXiv:2110.09456.

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

Mirac Suzgun, Yonatan Belinkov, Stuart Shieber, and Sebastian Gehrmann. 2019. LSTM networks can perform dynamic counting. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 44–54, Florence. Association for Computational Linguistics.

Anej Svete and Ryan Cotterell. 2024. Transformers can represent $n$-gram language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6845–6881, Mexico City, Mexico. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Ke Tran, Arianna Bisazza, and Christof Monz. 2018. The importance of being recurrent for modeling hierarchical structure. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4731–4736, Brussels, Belgium. Association for Computational Linguistics.

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

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

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

Kaiyue Wen, Yuchen Li, Bingbin Liu, and Andrej Risteski. 2023. Transformers are uninterpretable with myopic methods: a case study with bounded dyck grammars. In *Advances in Neural Information Processing Systems*, volume 36, pages 38723–38766. Curran Associates, Inc.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. 2020. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR.

Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. 2021. Self-attention networks can process bounded hierarchical languages. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3770–3785, Online. Association for Computational Linguistics.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

## A Additional Related Work

Since the emergence of Transformer (Vaswani et al., 2017), a wide range of theoretical analyses have been conducted on its expressive capacity. Some of these analyses focus on language recognition and generation tasks. These analyses can be broadly classified into two categories: (i) studies on the expressive capacity using circuit complexity and (ii) studies on the expressive power by examining specific languages.

### A.1 Theoretical analyses based on circuit complexity

There have been studies that try to identify the language classes that Transformers can process from the perspective of circuit complexity. Hao et al. (2022) established the relationship between unique hard attention Transformers (UHAT) and circuits, showing that UHAT can only recognize languages in the circuit class $\mathsf{AC}^0$, where $\mathsf{AC}^0$ is a circuit class that circuits consisting of constant depth and polynomial size AND and OR gates belong to. In addition, Barcelo et al. (2024) showed that UHAT cannot recognize all languages in $\mathsf{AC}^0$. In contrast, Hao et al. (2022), Merrill et al. (2022), Merrill and Sabharwal (2023), Strobl (2023), and Barcelo et al. (2024) provided theoretical results on saturated attention, or average hard attention (AHAT), which extends hardmax attention to be able to refer more than one token. Hao et al. (2022) showed that AHAT has strictly higher expressive power compared to UHAT. Merrill et al. (2022) provided a proof that AHAT can only recognize languages in the circuit class $\mathsf{TC}^0$, where $\mathsf{TC}^0$ is an extended circuit class of $\mathsf{AC}^0$ by adding majority gates to AND and OR gates. Barcelo et al. (2024) showed that AHAT can recognize languages within the linear temporal logic extended to require counting. Merrill and Sabharwal (2023) showed that log-precision Transformers can only recognize the languages within the class of uniform $\mathsf{TC}^0$. Strobl (2023) showed that AHAT can also recognize the languages within the class of uniform $\mathsf{TC}^0$.

Moreover, Li et al. (2024) showed that even with $O(\log n)$-steps chain-of-thought, Transformers with a $\mathrm{poly}(n)$-size representation can only process the languages within $\mathsf{AC}^0$ in case of constant-precision, and $\mathsf{TC}^0$ in case of $O(\log n)$-bit fixed point numbers, where $n$ is a sequence length and $\mathrm{poly}(n)$ is a polynomial function of $n$. In contrast, Li et al. (2024) showed that polynomial steps of chain-of-thought with respect to the sequence length enhance the expressive capacity of log-precision Transformers, showing that the problem class that log-precision Transformers with polynomial steps chain-of-thought can compute is equivalent to P/poly. Here, P/poly refers to the class of problems that can be solved by circuits with polynomial-size gates and is a superclass of the problem class P. In addition, Li et al. (2024) showed that by allowing $n$-step chain-of-thought, constant-precision Transformers with $\log n$-size representation can simulate any finite state automata, indicating that Transformers with chain-of-thought can express any regular languages.

### A.2 Theoretical analyses on specific languages

On the other hand, some studies have focused on specific languages to examine the expressive power of Transformers, particularly for the parity language within regular languages, the $\mathtt{Dyck}_k$ language within context-free languages, and the $\mathtt{Shuffle\text{-}Dyck}_k$ language. Hahn (2020) pointed out that Lipschitz-continuous Transformers cannot solve the parity task, $\mathtt{Dyck}_1$, and $\mathtt{Dyck}_2$ for arbitrary lengths. This is because when one character out of an input string of length $n$ is changed, the change in the output decays at $O(1/n)$, indicating that the performance of Transformers with restricted a Lipschitz constant approaches random guessing as the input length increases. Meanwhile, Yao et al. (2021) and Chiang and Cholak (2022) showed that the theoretical limitations presented by Hahn (2020) can be overcome by incorporating the layer normalization because the Lipschitz constant of the layer normalization can be $O(n)$. Chiang and Cholak (2022) also showed that Transformers with the layer normalization can solve the PARITY task by incorporating task-specific positional encoding $i/n$ and $(-1)^i$. Furthermore, Bhattamishra et al. (2020) theoretically showed that $O(k)$-width Transformers can recognize the $\mathtt{Shuffle\text{-}Dyck}_k$ language [1], suggesting that $O(k)$-width Transformers can process $k$ hierarchical structures in parallel.

In addition, there have also been studies that focus on how Transformers handle such hierarchical structures. Ebrahimi et al. (2020) focused on the Dyck language and demonstrated that the stack states appear in the attention patterns, suggesting

[1] Intuitively, the $\mathtt{Shuffle\text{-}Dyck}_k$ language is a set of strings composed of $k$ types of brackets, where all of the $k$ types of substrings are well-balanced. For instance, "([)]" belongs to $\mathtt{Shuffle\text{-}Dyck}_2$ not to $\mathtt{Dyck}_2$.

that the self-attention networks learn hierarchical structures within the attention layers. However, Wen et al. (2023) indicated that such attention patterns cannot be fully reliable. Moreover, Wen et al. (2023) provided a proof that a two-layer Transformer network with a width of $O(k^2D^2)$ can recognize $\texttt{Dyck}_{k,D}$. Furthermore, Yao et al. (2021) provided a constructive proof that by using specific absolute positional encoding $i/n$, 3-layer causal Transformers can recognize the $\texttt{Dyck}_{k,D}$ language. Yao et al. (2021) also showed that 2-layer causal Transformers with absolute positional encoding $i/n$, $i/n^3$, and $n$ can generate the $\texttt{Dyck}_k$ language.

### A.3 Analyses on the role of uninformative tokens

Moreover, there have been studies focusing on the importance of uninformative tokens — the BOS token in GPT (Radford et al., 2018) and the CLS and SEP tokens in BERT (Devlin et al., 2019). Clark et al. (2019), Devlin et al. (2019), and Kovaleva et al. (2019) observed that BERTs place relatively large attention on the CLS and SEP tokens. Clark et al. (2019) speculated that this phenomenon is for achieving "no-operations" within specific heads. In addition, Nye et al. (2021) observed that in algorithmic tasks, special tokens such as the CLS token serve as scratchpads, contributing to performance improvement.

In contrast, although the BOS token cannot refer to other tokens under causal masking, Ebrahimi et al. (2020) empirically showed that the presence of a starting token significantly improves the performance in recognizing the Dyck language. In addition, Weiss et al. (2021) showed that with a starting token, it is possible to determine how many tokens each head focuses on. Moreover, Kazemnejad et al. (2023) showed that with the BOS token, Transformers can create specific absolute and relative positional encoding. Furthermore, Xiao et al. (2024) demonstrated that regarding the Transformer architecture whose attention layers have restricted attention scope, by slightly modifying the architecture so that every token can refer to a starting token, the models perform significantly better. In light of these theoretical and empirical results, it has become evident that even tokens that do not have meaning by themselves are significant to enhance the performance of Transformers.

## B Detailed Preliminaries

We provide preliminaries for the proofs in the following sections and detailed definitions that are omitted due to the lack of space.

### B.1 $\texttt{Shuffle-Dyck}_k$

Following Suzgun et al. (2019), before defining the $\texttt{Shuffle-Dyck}_k$ language, we first recursively define the shuffling operation over two strings $\text{ш} : \Sigma^* \times \Sigma^* \to 2^{\Sigma^*}$ as follows:

$$u_1 \text{ш} \varepsilon = \varepsilon \text{ш} u_1 = \{u_1\}, \tag{20}$$

$$\begin{aligned}\beta_1 u_1 \text{ш} \beta_2 u_2 = \{\beta_1 u \mid u \in (u_1 \text{ш} \beta_2 u_2)\} \\ \cup \{\beta_2 u \mid u \in (\beta_1 u_1 \text{ш} u_2)\}\end{aligned} \tag{21}$$

for any $\beta_1, \beta_2 \in \Sigma$ and $u_1, u_2 \in \Sigma^*$. For instance,

$$\begin{aligned}&\langle_1 \rangle_1 \text{ш} \langle_2 \rangle_2 \\ &= \{\langle_1 \rangle_1 \langle_2 \rangle_2, \langle_1 \langle_2 \rangle_1 \rangle_2, \langle_1 \langle_2 \rangle_2 \rangle_1, \\ &\quad \langle_2 \rangle_2 \langle_1 \rangle_1, \langle_2 \langle_1 \rangle_2 \rangle_1, \langle_2 \langle_1 \rangle_1 \rangle_2\}.\end{aligned} \tag{22}$$

Moreover, we define the shuffling operation over $k$ strings $u_1, \cdots, u_k \in \Sigma^*$ and over $k$ languages $\mathcal{L}_1, \cdots, \mathcal{L}_k \subset \Sigma^*$ as follows:

$$\text{Ш}_{t=1}^{k} u_t = \bigcup_{u \in \text{Ш}_{t=1}^{k-1} u_t} u_k \text{ш} u, \tag{23}$$

$$\text{Ш}_{t=1}^{k} \mathcal{L}_t = \bigcup_{u_1 \in \mathcal{L}_1, \cdots, u_k \in \mathcal{L}_k} \text{Ш}_{t=1}^{k} u_t, \tag{24}$$

where $\text{Ш}_{t=1}^{1} u_t = \{u_1\}$.

**Definition 9** ($\texttt{Shuffle-Dyck}_k$ language for language models)**.** *The $\texttt{Shuffle-Dyck}_k$ language for language models is a language over an alphabet $\Sigma = \{\langle_t, \rangle_t\}_{t=1}^{k} \cup \{\texttt{<bos>}, \texttt{<eos>}\}$.*

*Given distinct $k$ $\texttt{Dyck}_1$ languages — $\texttt{Dyck}_1^1, \cdots, \texttt{Dyck}_1^k$, where $\texttt{Dyck}_1^t$ is the $\texttt{Dyck}_1$ language over an alphabet $\{\langle_t, \rangle_t\}$ —, the $\texttt{Shuffle-Dyck}_k$ language for language models is defined as follows:*

$$\left\{ \texttt{<bos>} w \texttt{<eos>} \;\middle|\; w \in \text{Ш}_{t=1}^{k} \texttt{Dyck}_1^t \right\} \tag{25}$$

Intuitively, the $\texttt{Shuffle-Dyck}_k$ language is a mixture of the $k$ $\texttt{Dyck}_1$ languages, and the ability to process the $\texttt{Shuffle-Dyck}_k$ language suggests that $k$ hierarchical structures can be processed in parallel. Figure 4 shows an example string that belongs to $\texttt{Shuffle-Dyck}_3$.

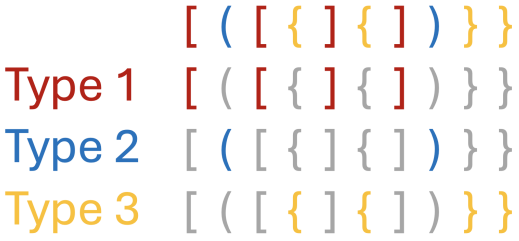


Figure 4: An example of string that belongs to $\texttt{Shuffle-Dyck}_3$ not to $\texttt{Dyck}_3$. Each substring of type $t \in \{1, 2, 3\}$ is properly balanced.

**Definition 10** ($\texttt{Shuffle-Dyck}_k$ language generation process)**.** *A language generation process* $p(w_{i+1} \mid w_{0:i})$ *over an alphabet* $\Sigma = \{\langle_t, \rangle_t\}_{t=1}^k \cup \{\texttt{<bos>}, \texttt{<eos>}\}$ *is called the* $\texttt{Shuffle-Dyck}_k$ *language generation process if*

$$p(w_0 = \text{“<bos>”} \mid \varepsilon) = 1, \tag{26}$$

$$p(w_{i+1} \mid w_{0:i}) = \begin{cases} p_0(w_{i+1}) & \text{if } \forall t \in [k].\, \mathrm{d}(w_{0:i} \mid t) = 0 \\ p_1(w_{i+1}) & \text{otherwise} \end{cases}, \tag{27}$$

*where* $\mathrm{d}(w_{0:i}|t)$ *represents the depth of the substring of type* $t$ *extracted from* $w_{0:i}$*, and* $p_0(w_{i+1}), p_1(w_{i+1})$ *are defined as follows:*

$$p_0(w_{i+1}) = \begin{cases} r\pi_t & \text{if } w_{i+1} = \text{“}\langle_t\text{”} \\ 1 - r & \text{if } w_{i+1} = \text{“<eos>”} \\ 0 & \text{otherwise} \end{cases}, \tag{28}$$

$$p_1(w_{i+1}) = \begin{cases} \frac{q\pi_t}{Z} & \text{if } w_{i+1} = \text{“}\langle_t\text{”} \\ \frac{(1-q)\overline{\pi}_t}{Z} & \text{if } w_{i+1} = \text{“}\rangle_t\text{”} \\ & \quad \wedge\ \mathrm{d}(w_{0:i} \mid t) > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{29}$$

*where* $\boldsymbol{\pi}, \overline{\boldsymbol{\pi}} \in \Delta^{k-1}$*, and*

$$Z = \sum_{t'=1}^{k} q\pi_{t'} + \sum_{t' \in \{t \mid \mathrm{d}(w_{0:i}|t) > 0\}} (1-q)\overline{\pi}_{t'}. \tag{30}$$

*Hereafter, we explicitly write the* $\texttt{Shuffle-Dyck}_k$ *language generation process parameterized by* $q, r, \boldsymbol{\pi}, \overline{\boldsymbol{\pi}}$ *as* $p_{\texttt{Shuffle-Dyck}_k}(\cdot; q, r, \boldsymbol{\pi}, \overline{\boldsymbol{\pi}})$.

### B.2 Transformer Architecture

We largely follow the Transformer architecture adopted in Yao et al. (2021); that is, we consider Transformer architecture composed of multiple Transformer blocks, each of which incorporates a self-attention layer and a feed-forward network layer.

Let $L$ be the number of Transformer blocks, $d_{\text{model}}$ be the dimension of the embedding vectors and hidden representations, $\Sigma$ be the vocabulary set, and $K$ be the vocabulary size.

Given an input string $w_{0:n} (= \texttt{<bos>} w_{1:n}) \in \Sigma^*$, which we identify with the sequence of one-hot vectors $\begin{bmatrix} \mathbf{e}_{w_0} & \cdots & \mathbf{e}_{w_n} \end{bmatrix} \in \mathbb{R}^{K \times (n+1)}$, the architecture process the string as follows:

$$\mathbf{x}_i^{(1)} = W_{\text{emb}} \mathbf{e}_{w_i} + \mathbf{p}_i, \tag{31}$$

$$\mathbf{h}_i^{(\ell)} = \mathrm{Att}\left(W_Q^{(\ell)} \mathbf{x}_i^{(\ell)}, W_K^{(\ell)} \mathbf{x}_{0:i}^{(\ell)}, W_V^{(\ell)} \mathbf{x}_{0:i}^{(\ell)}\right), \tag{32}$$

$$\mathbf{x}_i^{(\ell+1)} = \mathrm{FFN}\left(\mathbf{h}_i^{(\ell)}; W_1^{(\ell)}, W_2^{(\ell)}, \boldsymbol{\beta}^{(\ell)}, \boldsymbol{\gamma}^{(\ell)}\right), \tag{33}$$

where

- $\mathbf{x}_i^{(\ell)} \in \mathbb{R}^{d_{\text{model}}}$ is the $i$-th input representation to the $\ell$-th layer,
- $W_{\text{emb}} \in \mathbb{R}^{d_{\text{model}} \times K}$ is a linear embedding function,
- $\mathbf{p}_i \in \mathbb{R}^{d_{\text{model}}}$ is the positional encoding at the position $i$,
- $\mathrm{Att}(\cdot)$ is an attention layer, which is parameterized by three matrices $W_Q^{(\ell)}, W_K^{(\ell)}, W_V^{(\ell)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$,
- $\mathrm{FFN}(\cdot)$ is a feed-forward network layer, which is parameterized by two matrices $W_1^{(\ell)}, W_2^{(\ell)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ and $\boldsymbol{\beta}^{(\ell)}, \boldsymbol{\gamma}^{(\ell)} \in \mathbb{R}^{d_{\text{model}}}$.

Next, we describe the details of the attention and feed-forward network layers.

#### Attention layer

We consider attention layers with causal masking and the residual connection (He et al., 2015). Specifically, for $\ell$-th layer, the input sequence of length $n+1$ — $\mathbf{x}_0^{(\ell)}, \cdots, \mathbf{x}_n^{(\ell)}$ — is first processed with three token-wise linear transformations $W_Q^{(\ell)}, W_K^{(\ell)}, W_V^{(\ell)}$, which create $3(n+1)$ vectors

$\left\{W_Q^{(\ell)}\mathbf{x}_i^{(\ell)}, W_K^{(\ell)}\mathbf{x}_i^{(\ell)}, W_V^{(\ell)}\mathbf{x}_i^{(\ell)}\right\}_{i=0}^{n}$. Then, the $i$-th output $\mathbf{h}_i^{(\ell)}$ is calculated as follows:

$$\boldsymbol{\alpha}_i^{(\ell)} = \mathbb{S}\left(\left\langle W_Q^{(\ell)}\mathbf{x}_i^{(\ell)}, W_K^{(\ell)}\mathbf{x}_0^{(\ell)}\right\rangle, \cdots, \left\langle W_Q^{(\ell)}\mathbf{x}_i^{(\ell)}, W_K^{(\ell)}\mathbf{x}_i^{(\ell)}\right\rangle\right), \tag{34}$$

$$\mathbf{a}_i^{(\ell)} = \sum_{j=0}^{i} \alpha_{i,j}^{(\ell)} W_V^{(\ell)}\mathbf{x}_j^{(\ell)}, \tag{35}$$

$$\begin{aligned}\mathbf{h}_i^{(\ell)} &= \mathrm{Att}\left(W_Q^{(\ell)}\mathbf{x}_i^{(\ell)}, W_K^{(\ell)}\mathbf{x}_{0:i}^{(\ell)}, W_V^{(\ell)}\mathbf{x}_{0:i}^{(\ell)})\right) \\ &= \mathbf{x}_i^{(\ell)} + \mathbf{a}_i^{(\ell)},\end{aligned} \tag{36}$$

where $\langle\cdot,\cdot\rangle$ is a dot-product and $\mathbb{S}(\cdot)$ is a softmax operation.

**Feed-forward network layer**

A feed-forward network layer is a token-wise transformation that maps $\mathbf{h}_i \mapsto \mathrm{FFN}(\mathbf{h}_i)$. In this paper, we implement a feed-forward network as two linear transformations with the ReLU activations. We adopt the residual connection (He et al., 2015) and the RMS layer normalization (Zhang and Sennrich, 2019). This architecture largely follows that proposed in Yao et al. (2021) with a slight modification: we replace the standard layer normalization (Ba et al., 2016) with the RMS layer normalization (Zhang and Sennrich, 2019). Specifically, the feed-forward network transforms the vector $\mathbf{h}_i^{(\ell)}$ as follows:

$$\begin{aligned}&\mathrm{FFN}\left(\mathbf{h}_i^{(\ell)}; W_1^{(\ell)}, W_2^{(\ell)}, \boldsymbol{\beta}^{(\ell)}, \boldsymbol{\gamma}^{(\ell)}\right) \\ &= \mathbf{h}_i^{(\ell)} + W_2^{(\ell)}\left[\left(\mathrm{LN}_{\mathrm{RMS}}\left(W_1^{(\ell)}\mathbf{h}_i^{(\ell)}\right)\right)\right]_+,\end{aligned} \tag{37}$$

where $[\cdot]_+$ is a ReLU activation and $\mathrm{LN}_{\mathrm{RMS}}(\cdot)$ is the RMS layer normalization (Zhang and Sennrich, 2019) parameterized by $\boldsymbol{\beta}^{(\ell)}, \boldsymbol{\gamma}^{(\ell)} \in \mathbb{R}^{d_{\mathrm{model}}}$. Specifically,

$$\mathrm{LN}_{\mathrm{RMS}}(\mathbf{y}) = \boldsymbol{\gamma}^{(\ell)} \odot \frac{\mathbf{y}}{\mathrm{RMS}(\mathbf{y})} + \boldsymbol{\beta}^{(\ell)}, \tag{38}$$

where $\odot$ is an element-wise multiplication and

$$\mathrm{RMS}(\mathbf{y}) = \sqrt{\frac{1}{d_{\mathrm{model}}}\sum_{d=1}^{d_{\mathrm{model}}} y_d^2}. \tag{39}$$

Zhang and Sennrich (2019) empirically showed that the RMS layer normalization reduces the training time compared to the conventional layer normalization while maintaining their performances. The RMS layer normalization has been adopted in recent models such as Llama (Touvron et al., 2023a) and Llama 2 (Touvron et al., 2023b).

## C Notation

The notations used in this paper are summarized in Table 3.

## D Vector Representation

We define the vector representation that is used in the following sections. Specifically, the vector representation of the alphabet $\Sigma = \{\langle_t, \rangle_t\}_{t=1}^{k} \cup \{\texttt{<bos>}, \texttt{<eos>}\}$ takes the following form:

$$\begin{aligned}&\mathbf{x}\left(\in \mathbb{R}^{d_{\mathrm{model}}}\right) \\ &= \begin{bmatrix}\mathbf{t} \\ o \\ s \\ 1 \\ \mathbf{0}\end{bmatrix} \begin{matrix}\}\lceil \log_2 k\rceil \text{ dim.} \\ \}1 \text{ dim.} \\ \}1 \text{ dim.} \\ \}1 \text{ dim.} \\ \}(d_{\mathrm{model}} - \lceil \log_2 k\rceil - 3) \text{ dim.}\end{matrix},\end{aligned} \tag{40}$$

where

- $\mathbf{t} \in \{-1, 1\}^{\lceil \log_2 k\rceil} \cup \{\mathbf{0}\}$ represents a bracket-type embedding and $\mathbf{t}_{(t)}$ represents bracket-type embedding of type-$t$. Here, bracket types are encoded by $\pm 1$ binary encoding; for instance, when $k = 4$, 4 types are encoded into $\mathbf{t}_{(1)} = \begin{bmatrix}-1\\-1\end{bmatrix}, \mathbf{t}_{(2)} = \begin{bmatrix}-1\\1\end{bmatrix}, \mathbf{t}_{(3)} = \begin{bmatrix}1\\-1\end{bmatrix}, \mathbf{t}_{(4)} = \begin{bmatrix}1\\1\end{bmatrix}$. Note that the bracket-type embedding $\mathbf{t}$ of the two special tokens {"<bos>", "<eos>"} are defined as $\mathbf{0}$.
- $o \in \{-1, 0, 1\}$ represents the openness: $o = 1$ for open brackets, $o = -1$ for closed brackets, and $o = 0$ for two special tokens {"<bos>", "<eos>"}.
- $s \in \{0, 1\}$ is a starting signal that indicates whether the token is the starting token "<bos>" or not. This value is set to 1 for "<bos>" and 0 for the other tokens.
- $\mathbf{0} \in \mathbb{R}^{d_{\mathrm{model}} - \lceil \log_2 k\rceil - 3}$ denotes a zero vector. These dimensions are used as a memory and a scratchpad.

These vector representations are implemented with the following embedding matrix:

| Variable | Definition |
|---|---|
| $k$ | Number of bracket types |
| $\texttt{Dyck}_k$ / $\texttt{Shuffle-Dyck}_k$ | The Dyck / Shuffle-Dyck language with $k$ types of bracket pairs |
| $D$ | Maximum depth of the Dyck language |
| $\texttt{Dyck}_{k,D}$ | The $\texttt{Dyck}_k$ language with bounded depth $D$ |
| $\Sigma$ / $K$ | Vocaburary set / Vocaburary size |
| $t$ | Bracket type |
| "$\langle_t$" / "$\rangle_t$" | Open / Closed bracket of type $t$ |
| "<bos>" / "<eos>" / $\varepsilon$ | BOS / EOS token / Empty string |
| ш / Ш | Shuffling operation over two strings / multiple strings or languages |
| $n$ / $n_{\max}$ | Length of input string / Maximum length of the training dataset |
| $i, j$ | Index of position |
| $w_{0:i} (= \texttt{<bos>} w_{1:i})$ | Prefix of string $w_{0:n}$ with a length of $i+1$ |
| $\mathcal{L}$ | Language |
| $\mathrm{d}(\cdot)$ | Depth function |
| $p_{\mathcal{L}}(\cdot)$ | Language generation process of language $\mathcal{L}$ (Definition 6) |
| $q, r, \boldsymbol{\pi}$ / $q, r, \boldsymbol{\pi}, \overline{\boldsymbol{\pi}}$ | Parameters of the $\texttt{Dyck}_k$ / $\texttt{Shuffle-Dyck}_k$ language generation process |
| $L$ | Number of Transformer blocks |
| $d_{\text{model}}$ | Dimension of token representation |
| $W_{\text{emb}}$ | Token embedding matrix |
| $\mathbf{p}_i$ | Positional encoding at position $i$ |
| $\mathbf{x}_i^{(\ell)}$ | Input vector to the $\ell$-th layer at position $i$ |
| $\mathbf{h}_i^{(\ell)} \left(= \mathbf{x}_i^{(\ell)} + \mathbf{a}_i^{(\ell)}\right)$ | Output vector of the $\ell$-th attention layer at position $i$ |
| $\mathbf{t}_i$ / $o_i$ / $s_i$ | Bracket-type embedding / Openness of bracket / Starting signal |
| $\hat{s}$ | Pseudo starting signal |
| $\mathrm{Att}(\cdot)$ / $\mathrm{FFN}(\cdot)$ | Self-attention layer / Feed-forward network layer |
| $W_Q^{(\ell)}$ / $W_K^{(\ell)}$ / $W_V^{(\ell)}$ | Query / key / value matrices that parameterize $\mathrm{Att}(\cdot)$ in $\ell$-th layer |
| $W_1^{(\ell)}$ / $W_2^{(\ell)}$ | Weights of the first / second linear transformation in $\mathrm{FFN}(\cdot)$ in $\ell$-th layer |
| $\boldsymbol{\alpha}_i^{(\ell)}$ | Attention weights of query at position $i$ in $\ell$-th attention layer |
| $\mathrm{LN}(\cdot)$ / $\mathrm{LN}_{\mathrm{RMS}}(\cdot)$ | The layer normalization / The RMS layer normalization |
| $\mathrm{RMS}(\cdot)$ | Root mean square |
| $\boldsymbol{\beta}^{(\ell)}, \boldsymbol{\gamma}^{(\ell)}$ | Parameters of the RMS layer normalization in $\ell$-th attention layer |
| $\langle \cdot, \cdot \rangle$ / $\mathbb{S}(\cdot)$ / $\Delta^{K-1}$ | Dot product / Softmax function / $(K-1)$-dimensional probability simplex |
| $\mathcal{T}$ | Transformer $\Sigma^* \to \mathbb{R}^{* \times d_{\text{model}}}$, where $*$ represents an arbitrary length. |
| $f_{\text{rec}}$ / $f_{\text{gen}}$ | Recognizer head $\mathbb{R}^{d_{\text{model}}} \to \mathbb{R}$ / Generator head $\mathbb{R}^{d_{\text{model}}} \to \mathbb{R}^K$ |
| $\mathrm{sgn}(\cdot)$ | Sign function $\mathbb{R} \to \{1, -1\}$ |
| $p$ | Probability distribution over strings |
| $(\Sigma^*, \mathcal{F}, P)$ / $(\Sigma^*, \mathcal{F}', P')$ | Probability space / Complete extension of $(\Sigma^*, \mathcal{F}', P')$ |
| $\{w_{1:n}\}$ | Singleton set of a string $w_{1:n}$ |
| $a$ | attention score on a starting token <bos> |
| $\phi(\cdot)$ / $\theta(\cdot)$ | Function that converts position / depth to the angle |
| $Q(w_{0:i})$ | Propositional variable that indicates $w_{0:i}$ is a prefix for the $\texttt{Dyck}_k$ language |
| $\mathrm{q}(w_{0:i})$ | Variable associated with the propositional variable $Q(w_{0:i})$ |
| $\epsilon$ | Small value |
| $I$ | Identity matrix |

Table 3: Table of notations.

$$
\begin{aligned}
&W_{\mathrm{emb}} \left( \in \mathbb{R}^{d_{\mathrm{model}} \times K} \right) \\
&= \begin{bmatrix}
\mathbf{t}_{(1)} & \cdots & \mathbf{t}_{(k)} & \mathbf{t}_{(1)} & \cdots & \mathbf{t}_{(k)} & \mathbf{0} & \mathbf{0} \\
1 & \cdots & 1 & -1 & \cdots & -1 & 0 & 0 \\
0 & \cdots & 0 & 0 & \cdots & 0 & 1 & 0 \\
1 & \cdots & 1 & 1 & \cdots & 1 & 1 & 1 \\
\mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\uparrow & & \uparrow & \uparrow & & \uparrow & \uparrow & \uparrow \\
\langle_1 & \cdots & \langle_k & \rangle_1 & \cdots & \rangle_k & \texttt{<bos>} & \texttt{<eos>}
\end{bmatrix}.
\end{aligned} \tag{41}
$$

In addition, denote the vector representation at position $i$ by

$$
\mathbf{x}_i = \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \mathbf{0} \end{bmatrix} \tag{42}
$$

in the subsequent sections.

## E Proof of Proposition 1

**Proposition 7** (Restatement of Proposition 1)**.** *For any language $\mathcal{L} \subset \Sigma^*$ over a finite alphabet $\Sigma$ and any probability distribution $p$ over $\mathcal{L}$, there exists a language generation process that produces the given probability distribution $p$. In other words, there exists a language generation process $p_{\mathcal{L}}(w_{i+1} \mid \texttt{<bos>}w_{1:i})$ such that for any string $w_{1:n} \in \mathcal{L}$,*

$$
p(w_{1:n}) = p_{\mathcal{L}}(\texttt{<bos>}w_{1:n}\texttt{<eos>}), \tag{43}
$$

*where*

$$
\begin{aligned}
&p_{\mathcal{L}}(\texttt{<bos>}w_{1:n}\texttt{<eos>}) \\
&\quad = p_{\mathcal{L}}(\texttt{<bos>}) \\
&\qquad \cdot \left( \prod_{i=1}^{n} p_{\mathcal{L}}(w_i \mid \texttt{<bos>}w_{1:i-1}) \right) \\
&\qquad \cdot p_{\mathcal{L}}(\texttt{<eos>} \mid \texttt{<bos>}w_{1:n}).
\end{aligned} \tag{44}
$$

*Proof.* We introduce a probability space to handle probabilities over the countably infinite set $\Sigma^*$. Given a finite alphabet $\Sigma$ and a probability space $(\Sigma^*, \mathcal{F}, P)$ over $\Sigma^*$, we can assume that for any $w_{1:n} \in \Sigma^*$ such that $p(w_{1:n}) > 0$, the singleton set $\{w_{1:n}\}$ belongs to $\mathcal{F}$. Here, there exists a unique minimal complete extension of the probability space $(\Sigma^*, \mathcal{F}', P')$, where for any string $w_{1:n} \in \Sigma^*$, the singleton set $\{w_{1:n}\} \in \mathcal{F}'$, indicating that $\mathcal{F}' = 2^{\Sigma^*}$. Therefore, any subset in $\Sigma^*$ is $\mathcal{F}'$-measurable.

Next, we define $\mathrm{Cyl}(w_{1:n})$ for a string $w_{1:n} \in \Sigma^*$ as follows:

$$
\mathrm{Cyl}(w_{1:n}) = \{w'_{1:n'} \mid n' \geq n \wedge w'_{1:n} = w_{1:n}\}. \tag{45}
$$

Intuitively, $\mathrm{Cyl}(w_{1:n})$ is a string set whose elements have $w_{1:n}$ as a prefix. Since $\{w_{1:n}\}$ and $\mathrm{Cyl}(w_{1:n})$ are $\mathcal{F}'$-measurable, we can calculate the probability measure $P'(\{w_{1:n}\})$ and $P'(\mathrm{Cyl}(w_{1:n}))$.

Then, the language generation process defined below corresponds to the probability distribution $p$.

$$
\begin{aligned}
&p_{\mathcal{L}}(\texttt{<bos>} \mid \varepsilon) = 1, \\
&p_{\mathcal{L}}(w_{i+1} \mid \texttt{<bos>}w_{1:i}) \\
&\quad = \begin{cases} p_{\mathcal{L}}^{\mathrm{pos}} & \text{if } P'(\mathrm{Cyl}(w_{1:i})) > 0 \\ p_{\mathcal{L}}^{\mathrm{null}} & \text{otherwise} \end{cases},
\end{aligned} \tag{46}
$$

where

$$
\begin{aligned}
&p_{\mathcal{L}}^{\mathrm{pos}}(w_{i+1} \mid \texttt{<bos>}w_{1:i}) \\
&\quad = \begin{cases} \dfrac{P'(\{w_{1:i}\})}{P'(\mathrm{Cyl}(w_{1:i}))} & \text{if } w_{i+1} = \texttt{<eos>} \\ \dfrac{P'(\mathrm{Cyl}(w_{1:i+1}))}{P'(\mathrm{Cyl}(w_{1:i}))} & \text{otherwise} \end{cases},
\end{aligned} \tag{47}
$$

$$
\begin{aligned}
&p_{\mathcal{L}}^{\mathrm{null}}(w_{i+1} \mid \texttt{<bos>}w_{1:i}) \\
&\quad = \begin{cases} 1 & \text{if } w_{i+1} = \texttt{<eos>} \\ 0 & \text{otherwise} \end{cases}.
\end{aligned} \tag{48}
$$

This is because, for any $w_{1:n} \in \Sigma^*$ such that $p(w_{1:n}) > 0$,

$$
\begin{aligned}
&p_{\mathcal{L}}(\texttt{<bos>}w_{1:n}\texttt{<eos>}) \\
&= p_{\mathcal{L}}(\texttt{<bos>}) \cdot \prod_{i=1}^{n} p_{\mathcal{L}}(w_i \mid \texttt{<bos>}w_{1:i-1}) \\
&\quad \cdot p_{\mathcal{L}}(\texttt{<eos>} \mid \texttt{<bos>}w_{1:n}) \\
&= \prod_{i=1}^{n} \frac{P'(\mathrm{Cyl}(w_{1:i}))}{P'(\mathrm{Cyl}(w_{1:i-1}))} \cdot \frac{P'(\{w_{1:n}\})}{P'(\mathrm{Cyl}(w_{1:n}))} \\
&= \frac{P'(\{w_{1:n}\})}{P'(\mathrm{Cyl}(\varepsilon))} \\
&= P'(\{w_{1:n}\}) \\
&= P(\{w_{1:n}\}) = p(w_{1:n}).
\end{aligned} \tag{49}
$$

□

## F Proof of Proposition 2

**Proposition 8** (Restatement of Proposition 2)**.** *For any length $n$ and $\mathtt{Dyck}_k$ language generation process $p_{\mathtt{Dyck}_k}(\cdot; q, r, \boldsymbol{\pi})$, there exists $\epsilon_n$ such that if*

$\boldsymbol{\pi} > 0$ *then*

$$p_{\mathsf{Dyck}_k}(\texttt{<bos>}w_{1:n}\texttt{<eos>}; q, r, \boldsymbol{\pi}) \begin{cases} \geq \epsilon_n & \text{if } w_{1:n} \in \mathsf{Dyck}_k \\ = 0 & \text{if } w_{1:n} \notin \mathsf{Dyck}_k \end{cases} \tag{50}$$

*holds.*

*Proof.* When $w_{1:n} \in \mathsf{Dyck}_k$,

$$\begin{aligned} &p_{\mathsf{Dyck}_k}(\texttt{<bos>}w_{1:n}\texttt{<eos>}) \\ &= p_{\mathsf{Dyck}_k}(\texttt{<eos>} \mid \texttt{<bos>}w_{1:n}) \\ &\quad \cdot p_{\mathsf{Dyck}_k}(\texttt{<bos>}w_{1:n}) \\ &\quad \vdots \\ &= p_{\mathsf{Dyck}_k}(\texttt{<eos>} \mid \texttt{<bos>}w_{1:n}) \\ &\quad \cdot \left( \prod_{j=1}^{n} p_{\mathsf{Dyck}_k}(w_j \mid \texttt{<bos>}w_{1:j-1}) \right) \\ &\quad \cdot p_{\mathsf{Dyck}_k}(\texttt{<bos>}) \\ &\geq (1-r) \cdot (\min\{r, 1-q, q\pi_{\min}\})^n \ (=: \epsilon_n) \end{aligned} \tag{51}$$

where $\pi_{\min} = \min(\{\pi_t | 1 \leq t \leq k\})$.

On the other hand, when $w_{1:n} \notin \mathsf{Dyck}_k$, either $\mathrm{d}(w_{1:n}) > 0$ or $w_{1:n}$ has some invalid prefixes. If $\mathrm{d}(w_{1:n}) > 0$, $p_{\mathsf{Dyck}_k}(\texttt{<eos>} \mid \texttt{<bos>}w_{1:n}) = 0$, indicating $p(\texttt{<bos>}w_{0:n}\texttt{<eos>}) = 0$. If $w_{1:n}$ has some incorrect prefixes, regarding the shortest prefix $w_{1:j}$, either $w_j$ is an invalid closed bracket or a token other than brackets. In both cases, $p_{\mathsf{Dyck}_k}(w_j \mid \texttt{<bos>}w_{1:j-1}) = 0$ holds, indicating $p_{\mathsf{Dyck}_k}(\texttt{<bos>}w_{1:n}\texttt{<eos>}) = 0$. □

## G Proof of Theorem 1

In this section, we present a constructive proof that Transformers without positional encoding can recognize the $\mathsf{Dyck}_k$ language using <bos>. We restate Theorem 1 for convenience.

**Theorem 3** (Restatement of Theorem 1, Transformers with a starting token, $\mathsf{Dyck}_k$ recognition)**.** *For all $k$, there exists a 5-layer $O(\log k)$-width causal Transformer without positional encoding that recognizes the $\mathsf{Dyck}_k$ language. Each layer incorporates both the residual connection and the layer normalization. This network is followed by a fully-connected layer and a sign function to output an acceptance signal.*

*Proof.* As shown in the proof sketch of Theorem 1, each layer performs the following operations. Figure 1 provides the high-level understanding of this Transformer. Note that $w_{0:i}$ corresponds to $\texttt{<bos>}w_{1:i}$.

**First layer** creates pseudo positional encoding $\begin{bmatrix}\cos\phi(i) & \sin\phi(i)\end{bmatrix}^\top$ at position $i$, where $\phi(i) = \tan^{-1}(i/\exp(a))$ and $a$ is an attention score on <bos>. Figure 5 provides the illustration of this layer.

**Second and third layers** count depth $\mathrm{d}(w_{0:i})$ and $\mathrm{d}(w_{0:i}) + 1$, respectively. This is because the depth of the closed bracket is smaller by 1 than the corresponding open bracket. For instance, the depths calculated for "$\langle_1\rangle_1$" are 1 for "$\langle_1$" and 0 for "$\rangle_1$". These computations are achieved by constructing a value matrix that outputs 1 for open brackets and $-1$ for closed brackets in a specific dimension.

**Fourth layer** calculates a value that corresponds to a propositional variable $Q(w_{0:i})$ that indicates $w_{1:i} \in \mathsf{Dyck}_k$ but is guaranteed to return the expected value ($w_{1:i} \in \mathsf{Dyck}_k$) only when $i = 0$ or $w_{1:i-1}$ is a prefix for $\mathsf{Dyck}_k$. Therefore, to check whether $w_{1:i} \in \mathsf{Dyck}_k$ in the subsequent layers, we have to check whether all propositional variables $\{Q(w_{0:j})\}_{j=0}^{i}$ return `True` or not.

**Fifth layer** calculates (i) whether $w_{1:n}$ is a prefix for $\mathsf{Dyck}_k$ with $\bigwedge_{i=1}^{n} Q(w_{0:i})$ and (ii) whether $\mathrm{d}(w_{0:i}) = 0$ or not.

We show the specific implementations for each layer in the subsequent subsections.

Note that we explicitly represent the layer number to which each variable or parameter belongs as a superscript. For instance, $W_V^{(2)}$ represents the value matrix that belongs to the second attention layer. In addition, we use concise notation $\mathrm{d}_i$ instead of $\mathrm{d}(w_{0:i})$. Moreover, we frequently use omitted representations for vectors or matrices, where the omitted dimensions of the transformation matrices are zero-padded. For instance, let

$$\mathbf{y}_i = \begin{bmatrix} \mathbf{a}_i \\ \mathbf{b}_i \\ c_i \\ \mathbf{0} \end{bmatrix} \begin{matrix} \}d_a \text{ dim.} \\ \}d_b \text{ dim.} \\ \}1 \text{ dim.} \\ \}d_0 \text{ dim.} \end{matrix} \tag{52}$$

be an example of an input vector. In this case, if

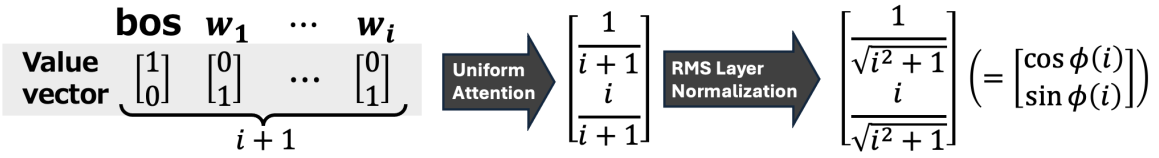


Figure 5: Intuitive illustration of the Transformer block that creates pseudo positional encoding $\begin{bmatrix}\cos\phi(i) & \sin\phi(i)\end{bmatrix}^\top$ at position $i$.

we use omitted representations

$$\mathbf{y}_i = \begin{bmatrix}\mathbf{a}_i\\ \vdots\\ c_i\\ \vdots\end{bmatrix}, \tag{53}$$

$$W = \begin{bmatrix}\mathbf{w}_{11}^\top & \cdots & w_{12} & \cdots\\ W_{21} & \cdots & \mathbf{w}_{22} & \cdots\\ \vdots & & \vdots & \end{bmatrix} \begin{matrix}\}1 \text{ dim.}\\ \}d_w \text{ dim.}\end{matrix}, \tag{54}$$

then, the matrix-vector product $W\mathbf{y}_i$ corresponds to the following computation:

$$\begin{aligned}
&W\mathbf{y}_i\\
&= \begin{bmatrix}\mathbf{w}_{11}^\top & \cdots & w_{12} & \cdots\\ W_{21} & \cdots & \mathbf{w}_{22} & \cdots\\ \vdots & & \vdots & \end{bmatrix} \begin{bmatrix}\mathbf{a}_i\\ \vdots\\ c_i\\ \vdots\end{bmatrix}\\
&= \begin{bmatrix}\mathbf{w}_{11}^\top & \mathbf{0}^\top & w_{12} & \mathbf{0}^\top\\ W_{21} & O & \mathbf{w}_{22} & O\\ O & O & \mathbf{0} & O\end{bmatrix} \begin{bmatrix}\mathbf{a}_i\\ \mathbf{b}_i\\ c_i\\ \mathbf{0}\end{bmatrix} \begin{matrix}\}d_a \text{ dim.}\\ \}d_b \text{ dim.}\\ \}1 \text{ dim.}\\ \}d_0 \text{ dim.}\end{matrix}\\
&\quad \underbrace{\phantom{\begin{bmatrix}\mathbf{w}_{11}^\top\\ W_{21}\\ O\end{bmatrix}}}_{d_a \text{ dim.}} \underbrace{\phantom{\begin{bmatrix}\mathbf{0}^\top\\ O\\ O\end{bmatrix}}}_{d_b \text{ dim.}} \underbrace{\phantom{\begin{bmatrix}w_{12}\\ \mathbf{w}_{22}\\ \mathbf{0}\end{bmatrix}}}_{1 \text{ dim.}} \underbrace{\phantom{\begin{bmatrix}\mathbf{0}^\top\\ O\\ O\end{bmatrix}}}_{d_0 \text{ dim.}}\\
&= \begin{bmatrix}\mathbf{w}_{11}^\top\mathbf{a}_i + w_{12}c_i\\ W_{21}\mathbf{a}_i + c_i\mathbf{w}_{22}\\ \mathbf{0}\end{bmatrix} \begin{matrix}\}1 \text{ dim.}\\ \}d_w \text{ dim..}\end{matrix}
\end{aligned} \tag{55}$$

### G.1 First layer

In the first layer, the following pseudo positional encoding is created.

$$\begin{bmatrix}\cos\phi(i)\\ \sin\phi(i)\end{bmatrix} \in \mathbb{R}^2, \tag{56}$$

where $\phi(i) = \tan^{-1}\left(\frac{i}{\exp(a)}\right)$ and $a \in \mathbb{R}$ is a constant.

**First layer —Attention layer**

We omit the unnecessary dimensions of input vector $\mathbf{x}_i^{(1)}$ in this layer as follows:

$$\mathbf{x}_i^{(1)} = \begin{bmatrix}\vdots\\ s_i\\ 1\\ \vdots\end{bmatrix}. \tag{57}$$

Set the parameters $W_Q^{(1)}, W_K^{(1)}, W_V^{(1)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ as follows:

$$W_Q^{(1)} = \begin{bmatrix}\cdots & 0 & 1 & \cdots\\ & \vdots & \vdots & \end{bmatrix}, \tag{58}$$

$$W_K^{(1)} = \begin{bmatrix}\cdots & a & 0 & \cdots\\ & \vdots & \vdots & \end{bmatrix}, \tag{59}$$

$$W_V^{(1)} = \begin{bmatrix} & \vdots & \vdots & \\ \cdots & 1 & 0 & \cdots\\ \cdots & -1 & 1 & \cdots\\ & \vdots & \vdots & \end{bmatrix}. \tag{60}$$

Then, we obtain

$$W_Q^{(1)}\mathbf{x}_{i_q}^{(1)} = \begin{bmatrix}1\\ \vdots\end{bmatrix}, \tag{61}$$

$$W_K^{(1)}\mathbf{x}_{i_k}^{(1)} = \begin{bmatrix}s_{i_k} \cdot a\\ \vdots\end{bmatrix}, \tag{62}$$

$$W_V^{(1)}\mathbf{x}_{i_k}^{(1)} = \begin{bmatrix}\vdots\\ s_{i_k}\\ 1 - s_{i_k}\\ \vdots\end{bmatrix}, \tag{63}$$

$$\left\langle W_K^{(1)}\mathbf{x}_{i_k}^{(1)}, W_Q^{(1)}\mathbf{x}_{i_q}^{(1)} \right\rangle = s_{i_k} \cdot a. \tag{64}$$

Therefore, $\mathbf{a}_i^{(1)}$ becomes

$$
\begin{aligned}
\mathbf{a}_i^{(1)} &= \frac{\exp(a)}{\exp(a)+i} W_V^{(1)} \mathbf{x}_0^{(1)} \\
&\quad + \sum_{j=1}^{i} \frac{1}{\exp(a)+i} W_V^{(1)} \mathbf{x}_j^{(1)} \\
&= \begin{bmatrix} \vdots \\ \frac{\exp(a)}{\exp(a)+i} \\ 0 \\ \vdots \end{bmatrix} + \sum_{j=1}^{i} \begin{bmatrix} \vdots \\ 0 \\ \frac{1}{\exp(a)+i} \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} \vdots \\ \frac{\exp(a)}{\exp(a)+i} \\ \frac{i}{\exp(a)+i} \\ \vdots \end{bmatrix}
\end{aligned} \tag{65}
$$

Finally, considering the residual connection, we obtain

$$
\begin{aligned}
\mathbf{h}_i^{(1)} &= \mathbf{x}_i^{(1)} + \begin{bmatrix} \vdots \\ \frac{\exp(a)}{\exp(a)+i} \\ \frac{i}{\exp(a)+i} \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \frac{\exp(a)}{\exp(a)+i} \\ \frac{i}{\exp(a)+i} \\ \mathbf{0} \end{bmatrix}.
\end{aligned} \tag{66}
$$

**First layer — Feed-forward network layer**

We omit the unnecessary dimensions of input vector $\mathbf{h}_i^{(1)}$ in this layer as follows:

$$
\mathbf{h}_i^{(1)} = \begin{bmatrix} \vdots \\ \frac{\exp(a)}{\exp(a)+i} \\ \frac{i}{\exp(a)+i} \\ \vdots \end{bmatrix}. \tag{67}
$$

Set the parameters $W_1^{(1)}, W_2^{(1)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ and $\boldsymbol{\beta}^{(1)}, \boldsymbol{\gamma}^{(1)} \in \mathbb{R}^{d_{\text{model}}}$ as follows:

$$
W_1^{(1)} = \begin{bmatrix} \cdots & 1 & 0 & \cdots \\ \cdots & 0 & 1 & \cdots \\ \cdots & \mathbf{0} & \mathbf{0} & \cdots \end{bmatrix}, \tag{68}
$$

$$
W_2^{(1)} = \begin{bmatrix} \vdots & \vdots & \vdots \\ 1 & 0 & \mathbf{0}^\top \\ 0 & 1 & \mathbf{0}^\top \\ \vdots & \vdots & \vdots \end{bmatrix}, \tag{69}
$$

$$
\boldsymbol{\beta}^{(1)} = \mathbf{0}, \tag{70}
$$

$$
\boldsymbol{\gamma}^{(1)} = \sqrt{\frac{1}{d_{\text{model}}}} \mathbf{1}. \tag{71}
$$

Then, the output of the FFN becomes

$$
\begin{aligned}
&W_2^{(1)} \left[ \mathrm{LN}_{\mathrm{RMS}} \left( W_1^{(1)} \mathbf{h}_i^{(1)} \right) \right]_+ \\
&= W_2^{(1)} \left[ \mathrm{LN}_{\mathrm{RMS}} \left( \begin{bmatrix} \frac{\exp(a)}{\exp(a)+i} \\ \frac{i}{\exp(a)+i} \\ \mathbf{0} \end{bmatrix} \right) \right]_+ \\
&= W_2^{(1)} \begin{bmatrix} \frac{\exp(a)}{\sqrt{\exp(a)^2+i^2}} \\ \frac{i}{\sqrt{\exp(a)^2+i^2}} \\ \mathbf{0} \end{bmatrix}_+ \\
&= \begin{bmatrix} \vdots & \vdots & \vdots \\ 1 & 0 & \mathbf{0}^\top \\ 0 & 1 & \mathbf{0}^\top \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \cos\phi(i) \\ \sin\phi(i) \\ \mathbf{0} \end{bmatrix} \\
&\quad \left( \text{from } \frac{\sin\phi(i)}{\cos\phi(i)} = \tan\phi(i) = \frac{i}{\exp(a)} \right) \\
&= \begin{bmatrix} \vdots \\ \cos\phi(i) \\ \sin\phi(i) \\ \vdots \end{bmatrix}.
\end{aligned} \tag{72}
$$

Finally, considering the residual connection, we obtain

$$
\begin{aligned}
\mathbf{x}_i^{(2)} &= \mathbf{h}_i^{(1)} + \begin{bmatrix} \vdots \\ \cos\phi(i) \\ \sin\phi(i) \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \frac{\exp(a)}{\exp(a)+i} \\ \frac{i}{\exp(a)+i} \\ \cos\phi(i) \\ \sin\phi(i) \\ \vdots \end{bmatrix}.
\end{aligned} \tag{73}
$$

### G.2 Second layer

In the second layer, the following vector that indicates the depth $\mathrm{d}(w_{0:i})$ is calculated:

$$\begin{bmatrix}\cos\theta(\mathrm{d}(w_{0:i}))\\ \sin\theta(\mathrm{d}(w_{0:i}))\end{bmatrix}, \tag{74}$$

where $\theta(\mathrm{d}) = \tan^{-1}\left(\frac{\mathrm{d}}{\exp(a)}\right)$.

**Second layer — Attention layer**

Recall that $o_i \in \{-1, 0, 1\}$ represents the openness of a bracket (1 for open brackets, $-1$ for closed brackets, and 0 for other tokens); that is, $\mathrm{d}(w_{0:i}) = \sum_{j=0}^{i} o_i$ holds.

We omit the unnecessary dimensions of input vector $\mathbf{x}_i^{(2)}$ in this layer as follows:

$$\mathbf{x}_i^{(2)} = \begin{bmatrix}\vdots\\ o_i\\ s_i\\ 1\\ \vdots\end{bmatrix}. \tag{75}$$

Set the parameters $W_Q^{(2)}, W_K^{(2)}, W_V^{(2)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ as follows:

$$W_Q^{(2)} = \begin{bmatrix}\cdots & 0 & 0 & 1 & \cdots\\ & \vdots & \vdots & \vdots & \end{bmatrix}, \tag{76}$$

$$W_K^{(2)} = \begin{bmatrix}\cdots & 0 & a & 0 & \cdots\\ & \vdots & \vdots & \vdots & \end{bmatrix}, \tag{77}$$

$$W_V^{(2)} = \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \cdots & 0 & 1 & 0 & \cdots\\ \cdots & 1 & 0 & 0 & \cdots\\ & \vdots & \vdots & \vdots & \end{bmatrix}. \tag{78}$$

Then, we obtain

$$W_Q^{(2)}\mathbf{x}_{i_q}^{(2)} = \begin{bmatrix}1\\ \vdots\end{bmatrix}, \tag{79}$$

$$W_K^{(2)}\mathbf{x}_{i_k}^{(2)} = \begin{bmatrix}s_{i_k}\cdot a\\ \vdots\end{bmatrix}, \tag{80}$$

$$W_V^{(2)}\mathbf{x}_{i_k}^{(2)} = \begin{bmatrix}\vdots\\ s_{i_k}\\ o_{i_k}\\ \vdots\end{bmatrix}, \tag{81}$$

$$\left\langle W_K^{(2)}\mathbf{x}_{i_k}^{(2)}, W_Q^{(2)}\mathbf{x}_{i_q}^{(2)} \right\rangle = s_{i_k}\cdot a. \tag{82}$$

Therefore, the output of the attention layer $\mathbf{a}_i^{(2)}$ becomes

$$\begin{aligned}\mathbf{a}_i^{(2)} &= \frac{\exp(a)}{\exp(a)+i}W_V^{(2)}\mathbf{x}_0^{(2)} + \sum_{j=1}^{i}\frac{1}{\exp(a)+i}W_V^{(2)}\mathbf{x}_j^{(2)}\\ &= \begin{bmatrix}\vdots\\ \frac{\exp(a)}{\exp(a)+i}\\ 0\\ \vdots\end{bmatrix} + \sum_{j=1}^{i}\begin{bmatrix}\vdots\\ 0\\ \frac{o_i}{\exp(a)+i}\\ \vdots\end{bmatrix}\\ &= \begin{bmatrix}\vdots\\ \frac{\exp(a)}{\exp(a)+i}\\ \frac{\mathrm{d}_i}{\exp(a)+i}\\ \vdots\end{bmatrix}\end{aligned} \tag{83}$$

Finally, considering the residual connection, we obtain

$$\begin{aligned}\mathbf{h}_i^{(2)} &= \mathbf{x}_i^{(2)} + \begin{bmatrix}\vdots\\ \frac{\exp(a)}{\exp(a)+i}\\ \frac{\mathrm{d}_i}{\exp(a)+i}\\ \vdots\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{t}_i\\ o_i\\ s_i\\ 1\\ \vdots\\ \cos\phi(i)\\ \sin\phi(i)\\ \frac{\exp(a)}{\exp(a)+i}\\ \frac{\mathrm{d}_i}{\exp(a)+i}\\ \mathbf{0}\end{bmatrix}\end{aligned} \tag{84}$$

**Second layer — Feed-forward network layer**

We omit the unnecessary dimensions of input vector $\mathbf{h}_i^{(2)}$ in this layer as follows:

$$\mathbf{h}_i^{(2)} = \begin{bmatrix}\vdots\\ \frac{\exp(a)}{\exp(a)+i}\\ \frac{\mathrm{d}_i}{\exp(a)+i}\\ \vdots\end{bmatrix}. \tag{85}$$

Set the parameters $W_1^{(2)}, W_2^{(2)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ and $\boldsymbol{\beta}^{(2)}, \boldsymbol{\gamma}^{(2)} \in \mathbb{R}^{d_{\text{model}}}$ as

follows:

$$W_1^{(2)} = \begin{bmatrix} \cdots & 1 & 0 & \cdots \\ \cdots & -1 & 0 & \cdots \\ \cdots & 0 & 1 & \cdots \\ \cdots & 0 & -1 & \cdots \\ & \vdots & \vdots & \end{bmatrix}, \tag{86}$$

$$W_2^{(2)} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \\ 1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & -1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}, \tag{87}$$

$$\boldsymbol{\beta}^{(2)} = \mathbf{0}, \tag{88}$$

$$\boldsymbol{\gamma}^{(2)} = \sqrt{\frac{2}{d_{\text{model}}}}\mathbf{1}. \tag{89}$$

Then, the output of the feed-forward network becomes

$$\begin{aligned} & W_2^{(2)} \left[ \mathrm{LN}_{\mathrm{RMS}} \left( W_1^{(2)} \mathbf{h}_i^{(2)} \right) \right]_+ \\ &= W_2^{(2)} \left[ \mathrm{LN}_{\mathrm{RMS}} \left( \begin{bmatrix} \frac{\exp(a)}{\exp(a)+i} \\ -\frac{\exp(a)}{\exp(a)+i} \\ \frac{\mathrm{d}_i}{\exp(a)+i} \\ -\frac{\mathrm{d}_i}{\exp(a)+i} \\ \vdots \end{bmatrix} \right) \right]_+ \\ &= \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \\ 1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & -1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix} \begin{bmatrix} \frac{\exp(a)}{\sqrt{\mathrm{d}_i^2 + \exp(a)^2}} \\ -\frac{\exp(a)}{\sqrt{\mathrm{d}_i^2 + \exp(a)^2}} \\ \frac{\mathrm{d}_i}{\sqrt{\mathrm{d}_i^2 + \exp(a)^2}} \\ -\frac{\mathrm{d}_i}{\sqrt{\mathrm{d}_i^2 + \exp(a)^2}} \\ \vdots \end{bmatrix}_+ \\ &= \begin{bmatrix} \vdots \\ \cos\theta(\mathrm{d}_i) \\ [\sin\theta(\mathrm{d}_i)]_+ - [-\sin\theta(\mathrm{d}_i)]_+ \\ \vdots \end{bmatrix} \\ & \left( \text{from } \frac{\sin\theta(\mathrm{d}_i)}{\cos\theta(\mathrm{d}_i)} = \tan\theta(\mathrm{d}_i) = \frac{\mathrm{d}_i}{\exp(a)} \right) \\ &= \begin{bmatrix} \vdots \\ \cos\theta(\mathrm{d}_i) \\ \sin\theta(\mathrm{d}_i) \\ \vdots \end{bmatrix} \end{aligned} \tag{90}$$

Finally, considering the residual connection, we obtain

$$\begin{aligned} \mathbf{x}_i^{(3)} &= \mathbf{h}_i^{(2)} + \begin{bmatrix} \vdots \\ \cos\theta(\mathrm{d}_i) \\ \sin\theta(\mathrm{d}_i) \\ \vdots \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \vdots \\ \cos\phi(i) \\ \sin\phi(i) \\ \vdots \\ \cos\theta(\mathrm{d}_i) \\ \sin\theta(\mathrm{d}_i) \\ \mathbf{0} \end{bmatrix}. \end{aligned} \tag{91}$$

### G.3 Third layer

The third layer counts depth $\mathrm{d}(w_{0:i})+1$ in addition to $\mathrm{d}(w_{0:i})$ which is counted in the second layer. This is because the depth of the closed bracket is smaller by 1 than the corresponding open bracket. For instance, the depths calculated for "$\langle_1\rangle_1$" are 1 for "$\langle_1$" and 0 for "$\rangle_1$".

The way to construct parameters is largely the same as that of the second layer. Specifically, we slightly modify the value matrix: we use

$$W_V^{(3)} = \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \cdots & 0 & 1 & 0 & \cdots \\ \cdots & 1 & \exp(-a) & 0 & \cdots \\ & \vdots & \vdots & \vdots & \end{bmatrix} \tag{92}$$

instead of

$$W_V^{(2)} = \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \cdots & 0 & 1 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & \cdots \\ & \vdots & \vdots & \vdots & \end{bmatrix}. \tag{93}$$

Then, we obtain

$$\begin{aligned}\mathbf{a}_i^{(3)} &= \frac{\exp(a)}{\exp(a)+i} W_V^{(3)} \mathbf{x}_0^{(3)} \\ &\quad + \sum_{j=1}^{i} \frac{1}{\exp(a)+i} W_V^{(3)} \mathbf{x}_j^{(3)} \\ &= \begin{bmatrix} \vdots \\ \frac{\exp(a)}{\exp(a)+i} \\ \frac{1}{\exp(a)+i} \\ \vdots \end{bmatrix} + \sum_{j=1}^{i} \begin{bmatrix} \vdots \\ 0 \\ \frac{o_i}{\exp(a)+i} \\ \vdots \end{bmatrix} \\ &= \begin{bmatrix} \vdots \\ \frac{\exp(a)}{\exp(a)+i} \\ \frac{\mathrm{d}_i+1}{\exp(a)+i} \\ \vdots \end{bmatrix}. \end{aligned} \tag{94}$$

Therefore, using the subsequent feed-forward network layer, we obtain

$$\mathbf{x}_i^{(4)} = \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \vdots \\ \cos\phi(i) \\ \sin\phi(i) \\ \vdots \\ \cos\theta(\mathrm{d}_i) \\ \sin\theta(\mathrm{d}_i) \\ \vdots \\ \cos\theta(\mathrm{d}_i+1) \\ \sin\theta(\mathrm{d}_i+1) \\ \mathbf{0} \end{bmatrix}. \tag{95}$$

### G.4 Fourth layer

The last two layers determine whether the input string belongs to the $\mathtt{Dyck}_k$ language, leveraging the position vectors and depth vectors computed so far. Note that the necessary and sufficient condition for a string $w_{1:n}$ to belong to the $\mathtt{Dyck}_k$ language is that the following two conditions are simultaneously satisfied.

**Condition (i)** $w_{1:n} \in \mathrm{Pre}(\mathtt{Dyck}_k)$.

**Condition (ii)** $\mathrm{d}(w_{1:n}) = 0$.

We then show how the two conditions above can be checked.

**Assessment of Condition (i)** We can check whether **Condition (i)** is satisfied by calculating $\bigwedge_{i=0}^{n} Q(w_{0:i})$, where $Q(w_{0:i})$ is a propositional variable that is guaranteed to return the expected boolean value ($w_{1:i} \in \mathrm{Pre}(\mathtt{Dyck}_k)$) only if $i = 0$ or $w_{1:i-1} \in \mathrm{Pre}(\mathtt{Dyck}_k)$. Specifically,

$$Q(w_{0:0}(=\texttt{<bos>})) = \mathtt{True}, \tag{96}$$

$$Q(w_{0:i}) = \begin{cases} \mathtt{True} \text{ or } \mathtt{False} & \text{if } w_{1:i-1} \notin \mathrm{Pre}(\mathtt{Dyck}_k) \\ \mathtt{True} & \text{else if } w_{1:i} \in \mathrm{Pre}(\mathtt{Dyck}_k) \\ \mathtt{False} & \text{else if } w_{1:i} \notin \mathrm{Pre}(\mathtt{Dyck}_k) \end{cases} \tag{97}$$

for $i \geq 1$.

Then we can check whether $w_{1:i}$ is a prefix for the Dyck language by calculating $\bigwedge_{i=0}^{n} Q(w_{0:i})$. We then explain that $\bigwedge_{i=0}^{n} Q(w_{0:i})$ precisely corresponds to whether $w_{1:i} \in \mathrm{Pre}(\mathtt{Dyck}_k)$.

**(i.1) If $\bigwedge_{i=0}^{n} Q(w_{0:i})$ is `True`.** In this case, since $Q(w_{0:i}) = \mathtt{True}$ holds for any $i$, it can be inductively shown that every propositional variable $Q(w_{0:i})$ precisely corresponds to $w_{1:i} \in \mathrm{Pre}(\mathtt{Dyck}_k)$; that is, $\bigwedge_{i=0}^{n} Q(w_{0:i}) = \mathtt{True}$ indicates $w_{1:n}$ is a prefix for $\mathtt{Dyck}_k$.

**(i.2) If $\bigwedge_{i=0}^{n} Q(w_{0:i})$ is `False`.** In this case, among the propositional variables that return `False`, with respect to the propositional variable at the smallest index $j$, all proposition variables preceding $Q(w_{0:j})$ return `True`; that is, it can be inductively shown that $w_{0:j-1} \in \mathrm{Pre}(\mathtt{Dyck}_k)$. Therefore, from the definition, $Q(w_{0:j}) = \mathtt{False}$ means $w_{1:j} \notin \mathrm{Pre}(\mathtt{Dyck}_k)$, indicating that $w_{1:n}$ is not a prefix for $\mathtt{Dyck}_k$.

**Assessment of Condition (ii)** We can easily check whether **Condition (ii)** is satisfied by checking whether $\sin\theta(\mathrm{d}_i) = 0$ or not.

As described above, it is possible to determine whether the input string $w_{1:n}$ is a prefix for the Dyck language or not using $\{Q(w_{0:i})\}_{i=0}^{n}$. Therefore, the fourth layer calculates the value $q(w_{0:i})$ that corresponds to the propositional variable $Q(w_{0:i})$.

#### Fourth layer — Attention layer

Recall that $\mathbf{t}_i \in \{-1, 1\}^{\lceil \log_2 k \rceil} \cup \{\mathbf{0}\}$ represents a bracket-type embedding. In the attention layer, each closed bracket at position $i$ fetches the bracket-type embedding $\mathbf{t}$ at the largest index among $\{0\} \cup$

$\{j \leq i \mid o_j = 1 \wedge \mathrm{d}_j = \mathrm{d}_i + 1\}$, while each open bracket fetches the bracket-type embedding of <bos>.

In a nutshell, the token at position $i$ fetches the bracket-type embedding $\tilde{\mathbf{t}}_i$, where $\tilde{\mathbf{t}}_i$ is the bracket-type embedding $\bar{\mathbf{t}}_i$ of the nearest depth-matched open bracket only if $w_i$ has depth-matched open brackets; otherwise, $\tilde{\mathbf{t}}_i = \mathbf{t}_0 (= \mathbf{0})$.

Before presenting the specific parameters, we first outline the method for calculating the attention scores when the query is a closed bracket in two steps: (i) assign high attention scores to the indices $\{0\} \cup \{j \leq i \mid o_j = 1 \wedge \mathrm{d}_j = \mathrm{d}_i + 1\}$; that is, extract a starting token and depth-matched open brackets and (ii) within those tokens, assign higher attention scores to tokens closer to the query, thereby focusing on the token with the largest index. Figure 6 illustrates this calculation, where the first step corresponds to the term $\mathrm{T}^{\mathrm{depth}}$ and the second step corresponds to the term $\mathrm{T}^{\mathrm{pos}}$.

We then show the specific parameters that achieve the desired operation. We omit the unnecessary dimensions of input vector $\mathbf{x}_i^{(4)}$ in this layer as follows:

$$\mathbf{x}_i^{(4)} = \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \vdots \\ \cos\phi(i) \\ \sin\phi(i) \\ \vdots \\ \cos\theta(\mathrm{d}_i) \\ \sin\theta(\mathrm{d}_i) \\ \vdots \\ \cos\theta(\mathrm{d}_i + 1) \\ \sin\theta(\mathrm{d}_i + 1) \\ \vdots \end{bmatrix}. \tag{98}$$

Set the parameters $W_Q^{(4)}, W_K^{(4)}, W_V^{(4)} \in \mathbb{R}^{d_{\mathrm{model}} \times d_{\mathrm{model}}}$ as follows (Note that in some cases, the transposed matrices are described to accommodate the limited space):

$$W_Q^{(4)} = \begin{bmatrix} C_1^{(4)} W_Q^{\mathrm{depth}} \\ W_Q^{\mathrm{pos}} \\ C_1^{(4)} \mathbf{w}_Q^{\mathrm{open}\top} \\ \vdots \end{bmatrix}, \tag{99}$$

$$W_K^{(4)} = \begin{bmatrix} C_2^{(4)} W_K^{\mathrm{depth}} \\ C_2^{(4)} W_K^{\mathrm{pos}} \\ C_2^{(4)} \mathbf{w}_K^{\mathrm{open}\top} \\ \vdots \end{bmatrix}, \tag{100}$$

$$W_V^{(4)\top} = \begin{bmatrix} \cdots & I & \cdots \\ \cdots & \mathbf{0}^\top & \cdots \\ \cdots & \mathbf{0}^\top & \cdots \\ \cdots & \mathbf{0}^\top & \cdots \\ & \vdots & \\ \cdots & \mathbf{0}^\top & \cdots \\ \cdots & \mathbf{0}^\top & \cdots \\ & \vdots & \\ \cdots & \mathbf{0}^\top & \cdots \\ \cdots & \mathbf{0}^\top & \cdots \\ & \vdots & \\ \cdots & \mathbf{0}^\top & \cdots \\ \cdots & \mathbf{0}^\top & \cdots \\ & \vdots & \end{bmatrix}, \tag{101}$$

where $C_1^{(4)}$ and $C_2^{(4)}$ are positive constants,

$$W_Q^{\mathrm{depth}\top} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}, \tag{102}$$

$$W_Q^{\mathrm{pos}\top} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 1 \\ -1 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \end{bmatrix}, \mathbf{w}_Q^{\mathrm{open}} = \begin{bmatrix} \mathbf{0} \\ 1 \\ -1 \\ 1 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \end{bmatrix}, \tag{103}$$

$$W_K^{\mathrm{depth}\top} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & -1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}, \tag{104}$$

$$W_K^{\mathrm{pos}\top} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \end{bmatrix}, \mathbf{w}_K^{\mathrm{open}} = \begin{bmatrix} \mathbf{0} \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \end{bmatrix}. \tag{105}$$

Then, we obtain

$$W_Q^{(4)}\mathbf{x}_{i_q}^{(4)} = \begin{bmatrix} C_1^{(4)} W_Q^{\mathrm{depth}} \mathbf{x}_{i_q}^{(4)} \\ W_Q^{\mathrm{pos}} \mathbf{x}_{i_q}^{(4)} \\ C_1^{(4)} \mathbf{w}_Q^{\mathrm{open}\top} \mathbf{x}_{i_q}^{(4)} \\ \vdots \end{bmatrix}, \tag{106}$$

$$W_K^{(4)}\mathbf{x}_{i_k}^{(4)} = \begin{bmatrix} C_2^{(4)} W_K^{\mathrm{depth}} \mathbf{x}_{i_k}^{(4)} \\ C_2^{(4)} W_K^{\mathrm{pos}} \mathbf{x}_{i_k}^{(4)} \\ C_2^{(4)} \mathbf{w}_K^{\mathrm{open}\top} \mathbf{x}_{i_k}^{(4)} \\ \vdots \end{bmatrix}, \tag{107}$$

$$W_V^{(4)}\mathbf{x}_{i_k}^{(4)} = \begin{bmatrix} \vdots \\ \mathbf{t}_{i_k} \\ \vdots \end{bmatrix}, \tag{108}$$

$$\begin{aligned} &\left\langle W_K^{(4)}\mathbf{x}_{i_k}^{(4)}, W_Q^{(4)}\mathbf{x}_{i_q}^{(4)} \right\rangle \\ &= C_2^{(4)} C_1^{(4)} \left\langle W_K^{\mathrm{depth}}\mathbf{x}_{i_k}^{(4)}, W_Q^{\mathrm{depth}}\mathbf{x}_{i_q}^{(4)} \right\rangle \\ &\quad + C_2^{(4)} \left\langle W_K^{\mathrm{pos}}\mathbf{x}_{i_k}^{(4)}, W_Q^{\mathrm{pos}}\mathbf{x}_{i_q}^{(4)} \right\rangle \\ &\quad + C_2^{(4)} C_1^{(4)} \mathbf{w}_K^{\mathrm{open}\top}\mathbf{x}_{i_k}^{(4)} \cdot \mathbf{w}_Q^{\mathrm{open}\top}\mathbf{x}_{i_q}^{(4)} \\ &= C_2^{(4)} \left( C_1^{(4)}\, \mathrm{T}_{i_q,i_k}^{\mathrm{depth}} + \mathrm{T}_{i_q,i_k}^{\mathrm{pos}} + C_1^{(4)}\, \mathrm{T}_{i_q,i_k}^{\mathrm{open}} \right), \end{aligned} \tag{109}$$

where

$$\begin{aligned} \mathrm{T}_{i_q,i_k}^{\mathrm{depth}} &= \left\langle W_K^{\mathrm{depth}}\mathbf{x}_{i_k}^{(4)}, W_Q^{\mathrm{depth}}\mathbf{x}_{i_q}^{(4)} \right\rangle \\ &= \cos(\theta(\mathrm{d}_{i_q}+1) - \theta(\mathrm{d}_{i_k})) \\ &\quad + \big(1 - \cos\theta(\mathrm{d}_{i_q}+1)\big)\big) \cdot s_{i_k} \\ &\quad + (o_{i_k} + s_{i_k} - 1) \\ &\begin{cases} = 1 & \text{if } w_{i_k} = \text{“<bos>”} \\ = 1 & \text{if } w_{i_k} = \text{“}\langle\text{.”} \\ & \quad \wedge\ d_{i_q} + 1 = d_{i_k} \\ < 1 & \text{otherwise} \end{cases}, \end{aligned} \tag{110}$$

$$\begin{aligned} \mathrm{T}_{i_q,i_k}^{\mathrm{pos}} &= \left\langle W_K^{\mathrm{pos}}\mathbf{x}_{i_k}^{(4)}, W_Q^{\mathrm{pos}}\mathbf{x}_{i_q}^{(4)} \right\rangle \\ &= -\sin(\phi(i_q) - \phi(i_k)), \end{aligned} \tag{111}$$

$$\begin{aligned} \mathrm{T}_{i_q,i_k}^{\mathrm{open}} &= \mathbf{w}_K^{\mathrm{open}\top}\mathbf{x}_{i_k}^{(4)} \cdot \mathbf{w}_Q^{\mathrm{open}\top}\mathbf{x}_{i_q}^{(4)} \\ &= (o_{i_q} - s_{i_q} + 1) \cdot s_{i_k} \\ &= \begin{cases} 2 & \text{if } w_{i_q} = \text{“}\rangle\text{.”} \\ & \quad \wedge\ w_{i_k} = \text{“<bos>”} \\ 0 & \text{otherwise} \end{cases}. \end{aligned} \tag{112}$$

Intuitively, $\mathrm{T}_{i_q,i_k}^{\mathrm{depth}}$ is a term that allows closed brackets to extract the depth-matched open brackets and <bos>, and $\mathrm{T}_{i_q,i_k}^{\mathrm{pos}}$ is a term that allows closed

brackets to extract the nearest token among them. Moreover, $\mathrm{T}^{\mathrm{open}}_{i_q,i_k}$ is a term that makes the query focus on the starting token only when the query is an open bracket. For example, the query "$\rangle_3$" in the input string "<bos>$\langle_2\langle_1\rangle_1\rangle_2\langle_3\rangle_3$" fetches the nearest depth-matched open bracket "$\langle_3$" as shown in Figure 6.

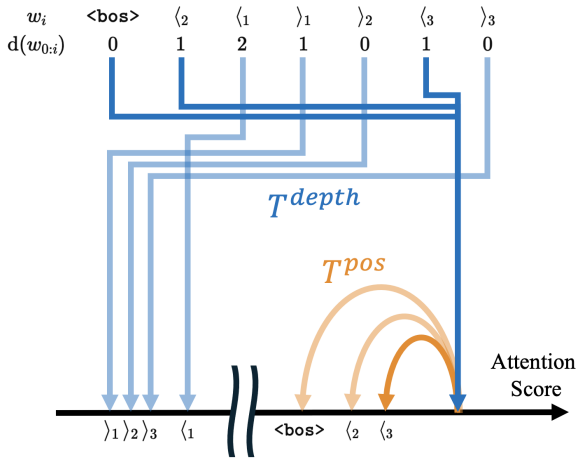


Figure 6: Illustration of the process where the query "$\rangle_3$" in the input string "<bos>$\langle_2\langle_1\rangle_1\rangle_2\langle_3\rangle_3$" fetches the nearest depth-matched open bracket "$\langle_3$". At first, using $\mathrm{T}^{\mathrm{depth}}$, only the depth-matched open brackets and <bos> are extracted, and then, using $\mathrm{T}^{\mathrm{pos}}$, the nearest one among them is extracted.

Therefore, when the query is a closed bracket, given a sufficiently large constant $C_1^{(4)}$ that satisfies $C_1^{(4)}(1-\cos(\theta(\mathrm{d}_{i_q}+1)-\theta(\mathrm{d}_{i_k})))>1$ if $\mathrm{d}_{i_q}+1\neq \mathrm{d}_{i_k}$,

$$\begin{aligned}
&\frac{1}{C_2^{(4)}}\left\langle W_K^{(4)}\mathbf{x}_{i_k}^{(4)}, W_Q^{(4)}\mathbf{x}_{i_q}^{(4)}\right\rangle \\
&= C_1^{(4)}\,\mathrm{T}^{\mathrm{depth}}_{i_q,i_k} + \mathrm{T}^{\mathrm{pos}}_{i_q,i_k} \\
&\begin{cases} = C_1^{(4)} - \sin(\phi(i_q)-\phi(i_k)) \\ \quad \text{if } w_{i_k} = \text{“<bos>”} \\ \quad \vee (w_{i_k} = \text{“}\langle\text{.”} \wedge \mathrm{d}_{i_q}+1=\mathrm{d}_{i_k}) \\ < C_1^{(4)} - 1 \\ \quad \text{otherwise} \end{cases}
\end{aligned} \tag{113}$$

holds, indicating that given a sufficiently large constant $C_2^{(4)}$, the query can focus on the nearest token among <bos> and depth-matched open brackets.

On the other hand, when the query is an open bracket,

$$\begin{aligned}
&\frac{1}{C_2^{(4)}}\left\langle W_K^{(4)}\mathbf{x}_{i_k}^{(4)}, W_Q^{(4)}\mathbf{x}_{i_q}^{(4)}\right\rangle \\
&= \begin{cases} C_1^{(4)}\,\mathrm{T}^{\mathrm{depth}}_{i_q,i_k} + \mathrm{T}^{\mathrm{pos}}_{i_q,i_k} + 2C_1^{(4)} \\ \quad \text{if } w_{i_k} = \text{“<bos>”} \\ C_1^{(4)}\,\mathrm{T}^{\mathrm{depth}}_{i_q,i_k} + \mathrm{T}^{\mathrm{pos}}_{i_q,i_k} \\ \quad \text{otherwise} \end{cases} \\
&\begin{cases} \geq 3C_1^{(4)} - 1 & \text{if } w_{i_k} = \text{“<bos>”} \\ \leq C_1^{(4)} & \text{otherwise} \end{cases}
\end{aligned} \tag{114}$$

holds, indicating that given a sufficiently large constant $C_1^{(4)}$, the query can focus on <bos>.

From the above, it is confirmed that the desired operations are performed correctly.

Thus, the output of the attention layer $\mathbf{a}_i^{(4)}$ becomes

$$\begin{aligned}
\mathbf{a}_i^{(4)} &= \sum_{j=0}^{i} \frac{1}{i+1} W_V^{(4)}\mathbf{x}_j^{(4)} \\
&= \begin{bmatrix} \vdots \\ \tilde{\mathbf{t}}_i \\ \vdots \end{bmatrix},
\end{aligned} \tag{115}$$

where $\tilde{\mathbf{t}}_i$ is the bracket-type embedding $\bar{\mathbf{t}}_i$ of the nearest depth-matched open bracket when $o_i=-1$ and $w_i$ has one or more such brackets; otherwise, it is set to the zero vector $\mathbf{0}$. Here, we treat softmax attention as hardmax attention for simplicity. However, as in Appendix O, it is sufficient if the attention allocated to the target token exceeds $\frac{2}{3}$ in practice.

Note that since we do not assume $\mathrm{d}_i \geq 0$, $\tilde{\mathbf{t}}_i$ is guaranteed to satisfy $\tilde{\mathbf{t}}_i \in \{-1,1\}^{\lceil \log_2 k\rceil} \cup \{\mathbf{0}\}$ even for strings that are not the prefixes for the Dyck language.

Finally, considering the residual connection, we

obtain

$$\mathbf{h}_i^{(4)} = \mathbf{x}_i^{(4)} + \begin{bmatrix} \vdots \\ \tilde{\mathbf{t}}_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \vdots \\ \cos\phi(i) \\ \sin\phi(i) \\ \vdots \\ \cos\theta(\mathrm{d}_i) \\ \sin\theta(\mathrm{d}_i) \\ \vdots \\ \cos\theta(\mathrm{d}_i+1) \\ \sin\theta(\mathrm{d}_i+1) \\ \tilde{\mathbf{t}}_i \\ \mathbf{0} \end{bmatrix}. \tag{116}$$

**Fourth layer — Feed-forward network layer**

In this layer, the objective is to compute $\mathrm{q}(w_{0:i})$, where $\mathrm{q}(w_{0:i})$ is positive when $Q(w_{0:i})$ is `True` and negative when $Q(w_{0:i})$ is `False`. Specifically,

$$\mathrm{q}(w_{0:i}) \begin{cases} > 1 & \text{if } o_i \neq -1 \\ > 1 & \text{if } o_i = -1 \wedge \mathbf{t}_i = \tilde{\mathbf{t}}_i \\ < -1 & \text{if } o_i = -1 \wedge \mathbf{t}_i \neq \tilde{\mathbf{t}}_i \end{cases}. \tag{117}$$

In the following proof, we use concise notation $\mathrm{q}_i$ instead of $\mathrm{q}(w_{0:i})$ and we omit the unnecessary dimensions of input vector $\mathbf{h}_i^{(4)}$ in this layer as follows:

$$\mathbf{h}_i^{(4)} = \begin{bmatrix} \mathbf{t}_i \\ o_i \\ \vdots \\ 1 \\ \vdots \\ \tilde{\mathbf{t}}_i \\ \vdots \end{bmatrix}. \tag{118}$$

Set the parameters $W_1^{(4)}, W_2^{(4)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ and $\boldsymbol{\beta}^{(4)}, \boldsymbol{\gamma}^{(4)} \in \mathbb{R}^{d_{\text{model}}}$ as follows:

$$W_1^{(4)} = \begin{bmatrix} I & \mathbf{0} & \cdots & \mathbf{0} & \cdots & -I & \cdots \\ -I & \mathbf{0} & \cdots & \mathbf{0} & \cdots & I & \cdots \\ \mathbf{0}^\top & 1 & \cdots & 1 & \cdots & \mathbf{0}^\top & \cdots \\ \mathbf{0}^\top & 0 & \cdots & 1 & \cdots & \mathbf{0}^\top & \cdots \\ \vdots & \vdots & & \vdots & & \vdots & \end{bmatrix}, \tag{119}$$

$$W_2^{(4)} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \\ -2\mathbf{1}^\top & -2\mathbf{1}^\top & C_3^{(4)} & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}, \tag{120}$$

$$\boldsymbol{\beta}^{(4)} = \mathbf{0}, \tag{121}$$

$$\boldsymbol{\gamma}^{(4)} = 8\sqrt{\frac{\lceil \log_2 k \rceil}{d_{\text{model}}}}\mathbf{1}, \tag{122}$$

where $C_3^{(4)} = 4(\lceil \log_2 k \rceil + 1)$. Then, we obtain

$$\begin{aligned} & W_2^{(4)} \left[ \mathrm{LN}_{\mathrm{RMS}} \left( W_1^{(4)} \mathbf{h}_i^{(4)} \right) \right]_+ \\ &= W_2^{(4)} \left[ \mathrm{LN}_{\mathrm{RMS}} \left( \begin{bmatrix} \mathbf{t}_i - \tilde{\mathbf{t}}_i \\ -(\mathbf{t}_i - \tilde{\mathbf{t}}_i) \\ o_i + 1 \\ 1 \\ \vdots \end{bmatrix} \right) \right]_+ \\ &= \begin{bmatrix} \vdots \\ \mathrm{q}_i \\ \vdots \end{bmatrix}, \end{aligned} \tag{123}$$

where

$$\begin{aligned} \mathrm{q}_i &= 8\sqrt{\frac{\lceil \log_2 k \rceil}{d_{\text{model}}}} \cdot \frac{1}{\mathrm{RMS}\left( W_1^{(4)} \mathbf{h}_i^{(4)} \right)} \\ &\quad \cdot \left( -2 \left\| \mathbf{t}_i - \tilde{\mathbf{t}}_i \right\|_1 + C_3^{(4)}(o_i + 1) + 1 \right) \\ &= 4\sqrt{\frac{\lceil \log_2 k \rceil}{2\|\mathbf{t}_i - \tilde{\mathbf{t}}_i\|_2^2 + (o_i+1)^2 + 1}} \cdot \mathrm{q}_i', \end{aligned} \tag{124}$$

$$\mathrm{q}_i' = -4 \left\| \mathbf{t}_i - \tilde{\mathbf{t}}_i \right\|_1 + 2C_3^{(4)}(o_i + 1) + 2. \tag{125}$$

Here

$$\begin{aligned} & 4\sqrt{\frac{\lceil \log_2 k \rceil}{2\|\mathbf{t}_i - \tilde{\mathbf{t}}_i\|_2^2 + (o_i+1)^2 + 1}} \\ &\geq 4\sqrt{\frac{\lceil \log_2 k \rceil}{2 \cdot 2^2 \lceil \log_2 k \rceil + 2^2 + 1}} \\ &\geq 4\sqrt{\frac{\lceil \log_2 k \rceil}{8\lceil \log_2 k \rceil + 8\lceil \log_2 k \rceil}} \\ &\geq 1 \end{aligned} \tag{126}$$

holds, indicating that it is sufficient to check $\mathrm{q}_i'$ satisfies the conditions instead of $\mathrm{q}_i$. We confirm the conditions by checking three patterns (i) $o_i \neq -1$; that is, $w_i$ is an open bracket or `<bos>`, (ii) $o_i = -1 \wedge \mathbf{t}_i = \tilde{\mathbf{t}}_i$; that is, $w_i$ is a closed bracket

and has a depth- and type-matched open bracket, and (iii) $o_i = -1 \wedge \mathbf{t}_i \neq \tilde{\mathbf{t}}_i$; that is, $w_i$ does not have the depth-matched brackets or has the depth-matchced brackets but faces type conflict regarding the closed bracket among them.

**(i) $w_i$ is an open bracket or <bos>.** In this case, since $o_i + 1 \geq 1$,

$$\begin{aligned} \mathrm{q}'_i &= -4 \left\| \mathbf{t}_i - \tilde{\mathbf{t}}_i \right\|_1 + 2C_3^{(4)}(o_i + 1) + 2 \\ &\geq -8\lceil \log_2 k \rceil + 2C_3^{(4)} + 2 \\ &= 10 > 1. \end{aligned} \tag{127}$$

**(ii) $w_i$ is a closed bracket and has a depth- and type-matched open bracket.** In this case, since $\tilde{\mathbf{t}}_i = \mathbf{t}_i$ and $o_i + 1 = 0$,

$$\begin{aligned} \mathrm{q}'_i &= -4 \left\| \mathbf{t}_i - \tilde{\mathbf{t}}_i \right\|_1 + 2C_3^{(4)}(o_i + 1) + 2 \\ &= 2 > 1. \end{aligned} \tag{128}$$

**(iii) $w_i$ is a closed bracket and faces a type conflict.** In this case, there are two exclusive subcases: (i) $w_i$ has no depth-matched open bracket; that is, $\tilde{\mathbf{t}}_i = \mathbf{0}$ holds and (ii) $w_i$ has depth-matched open brackets but faces type conflict; that is, $\tilde{\mathbf{t}}_i = \overline{\mathbf{t}}_i \neq \mathbf{t}_i$. In both subcases, $\|\tilde{\mathbf{t}}_i - \mathbf{t}_i\|_1 \geq 1$ and $o_i = -1$ hold; thus, we obtain

$$\begin{aligned} \mathrm{q}'_i &= -4 \left\| \mathbf{t}_i - \tilde{\mathbf{t}}_i \right\|_1 + 2C_3^{(4)}(o_i + 1) + 2 \\ &\leq -4 + 2 < -1. \end{aligned} \tag{129}$$

From the above, it is confirmed that the inequality (117) holds.

Finally, considering the residual connection, we obtain the following vectors:

$$\begin{aligned} \mathbf{x}_i^{(5)} &= \mathbf{h}_i^{(4)} + \begin{bmatrix} \vdots \\ \mathrm{q}_i \\ \vdots \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \vdots \\ \cos\phi(i) \\ \sin\phi(i) \\ \vdots \\ \cos\theta(\mathrm{d}_i) \\ \sin\theta(\mathrm{d}_i) \\ \vdots \\ \cos\theta(\mathrm{d}_i + 1) \\ \sin\theta(\mathrm{d}_i + 1) \\ \tilde{\mathbf{t}}_i \\ \mathrm{q}_i \\ \mathbf{0} \end{bmatrix}. \end{aligned} \tag{130}$$

### G.5 Fifth layer

The fifth layer check the two conditions: $\bigwedge_{i=0}^{n}(\mathrm{q}_i > 0)$ and $\mathrm{d}(w_{0:n}) = 0$.

**Fifth layer — Attention layer**

We omit the unnecessary dimensions of input vector $\mathbf{x}_i^{(5)}$ in this layer as follows:

$$\mathbf{x}_i^{(5)} = \begin{bmatrix} \vdots \\ s_i \\ 1 \\ \vdots \\ \mathrm{q}_i \\ \vdots \end{bmatrix}. \tag{131}$$

Set the parameters $W_Q^{(5)}, W_K^{(5)}, W_V^{(5)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ as follows:

$$W_Q^{(5)} = \begin{bmatrix} \cdots & 0 & C_1^{(5)} & \cdots & 0 & \cdots \\ \cdots & 0 & C_1^{(5)} & \cdots & 0 & \cdots \\ & \vdots & \vdots & & \vdots & \end{bmatrix}, \tag{132}$$

$$W_K^{(5)} = \begin{bmatrix} \cdots & 0 & 0 & \cdots & -1 & \cdots \\ \cdots & \mathrm{q}_0 & 0 & \cdots & 0 & \cdots \\ & \vdots & \vdots & & \vdots & \end{bmatrix}, \tag{133}$$

$$W_V^{(5)} = \begin{bmatrix} & \vdots & \vdots & & \vdots & \\ \cdots & -1 & 1 & \cdots & 0 & \cdots \\ & \vdots & \vdots & & \vdots & \end{bmatrix}, \tag{134}$$

where $C_1^{(5)}$ is a positive constant. Note that $\mathrm{q}_0$ can be treated as a constant because $\mathrm{q}_0 (= \mathrm{q}(\texttt{<bos>}))$ does not depend on the input string.

Then, we obtain

$$W_Q^{(5)} \mathbf{x}_{i_q}^{(5)} = \begin{bmatrix} C_1^{(5)} \\ C_1^{(5)} \\ \mathbf{0} \end{bmatrix}, \tag{135}$$

$$W_K^{(5)} \mathbf{x}_{i_k}^{(5)} = \begin{bmatrix} -\mathrm{q}_{i_k} \\ \mathrm{q}_0 \cdot s_{i_k} \\ \mathbf{0} \end{bmatrix}, \tag{136}$$

$$W_V^{(5)} \mathbf{x}_{i_k}^{(5)} = \begin{bmatrix} \mathbf{0} \\ 1 - s_{i_k} \\ \mathbf{0} \end{bmatrix}, \tag{137}$$

$$\begin{aligned} &\left\langle W_K^{(5)} \mathbf{x}_{i_k}^{(5)}, W_Q^{(5)} \mathbf{x}_{i_q}^{(5)} \right\rangle \\ &\quad = C_1^{(5)} \left( -\mathrm{q}_{i_k} + \mathrm{q}_0 \cdot s_{i_k} \right) \\ &\quad \begin{cases} = 0 & \text{if } i_k = 0 \\ < -C_1^{(5)} & \text{if } i_k \neq 0 \wedge \mathrm{q}_{i_k} > 0 \\ > C_1^{(5)} & \text{if } i_k \neq 0 \wedge \mathrm{q}_{i_k} < 0 \end{cases}. \end{aligned} \tag{138}$$

Intuitively, if $\{\mathrm{q}_{i_k}\}_{i_k=1}^{i_q}$ are all positive, attention scores on all tokens except on <bos> are much smaller than 0, making the query $\mathbf{x}_{i_q}$ focus on <bos>. In other words, the query $\mathbf{x}_{i_q}$ can focus on <bos> if and only if $w_{1:i_q}$ is a prefix for the $\texttt{Dyck}_k$ language. Therefore, given a sufficiently large constant $C_1^{(5)}$, the output of attention layer $\mathbf{a}_i^{(5)}$ becomes

$$\mathbf{a}_i^{(5)} = \begin{bmatrix} \mathbf{0} \\ \mathrm{q}_{\leq i} \\ \mathbf{0} \end{bmatrix}, \tag{139}$$

where

$$\mathrm{q}_{\leq i} = \begin{cases} 0 & \text{if } \forall j \in [i].\, \mathrm{q}_j > 0 \\ 1 & \text{if } \exists j \in [i].\, \mathrm{q}_j < 0 \end{cases} \tag{140}$$

Finally, considering the residual connection, we obtain the following vectors:

$$\begin{aligned} \mathbf{h}_i^{(5)} &= \mathbf{x}_i^{(5)} + \begin{bmatrix} \vdots \\ \mathrm{q}_{\leq i} \\ \vdots \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \vdots \\ \cos \phi(i) \\ \sin \phi(i) \\ \vdots \\ \cos \theta(\mathrm{d}_i) \\ \sin \theta(\mathrm{d}_i) \\ \vdots \\ \cos \theta(\mathrm{d}_i + 1) \\ \sin \theta(\mathrm{d}_i + 1) \\ \tilde{\mathbf{t}}_i \\ \mathrm{q}_i \\ \mathrm{q}_{\leq i} \\ \mathbf{0} \end{bmatrix}. \end{aligned} \tag{141}$$

Although we treat softmax attention as hardmax attention, it is sufficient that there exists a constant such that $\max\{\mathrm{q}_{\leq i} \mid \forall j \in [i].\, \mathrm{q}_j > 0\} < \min\{\mathrm{q}_{\leq i} \mid \exists j \in [i].\, \mathrm{q}_j < 0\}$, similar to the fourth layer.

**Fifth layer — Feed-forward network layer**

We omit the unnecessary dimensions of input vector $\mathbf{h}_i^{(5)}$ in this layer as follows:

$$\mathbf{h}_i^{(5)} = \begin{bmatrix} \vdots \\ \cos \theta(\mathrm{d}_i) \\ \sin \theta(\mathrm{d}_i) \\ \vdots \\ \mathrm{q}_{\leq i} \\ \vdots \end{bmatrix}. \tag{142}$$

Set the parameters $W_1^{(5)}, W_2^{(5)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ and $\boldsymbol{\beta}^{(5)}, \boldsymbol{\gamma}^{(5)} \in \mathbb{R}^{d_{\text{model}}}$ as follows:

$$W_1^{(5)} = \begin{bmatrix} \cdots & 0 & 0 & \cdots & 1 & \cdots \\ \cdots & 1 & 0 & \cdots & 0 & \cdots \\ \cdots & 0 & 1 & \cdots & 0 & \cdots \\ & \vdots & \vdots & & \vdots & \end{bmatrix}, \tag{143}$$

$$W_2^{(5)} = \begin{bmatrix} \vdots & \vdots & \vdots & \\ 1 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}, \tag{144}$$

$$\boldsymbol{\beta}^{(5)} = \mathbf{0}, \tag{145}$$

$$\boldsymbol{\gamma}^{(5)} = \sqrt{\frac{1}{d_{\text{model}}}}\mathbf{1}. \tag{146}$$

Then, the output of the feed-forward network layer becomes

$$\begin{aligned}
&W_2^{(5)}\left[\mathrm{LN}_{\mathrm{RMS}}\left(W_1^{(5)}\mathbf{h}_i^{(5)}\right)\right]_+ \\
&= W_2^{(5)}\left[\mathrm{LN}_{\mathrm{RMS}}\left(\begin{bmatrix} \mathrm{q}_{\leq i} \\ \cos\theta(\mathrm{d}_i) \\ \sin\theta(\mathrm{d}_i) \\ \vdots \end{bmatrix}\right)\right]_+ \\
&= W_2^{(5)}\begin{bmatrix} \frac{[\mathrm{q}_{\leq i}]_+}{\sqrt{1+\mathrm{q}_{\leq i}^2}} \\ \frac{[\cos\theta(\mathrm{d}_i)]_+}{\sqrt{1+\mathrm{q}_{\leq i}^2}} \\ \frac{[\sin\theta(\mathrm{d}_i)]_+}{\sqrt{1+\mathrm{q}_{\leq i}^2}} \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} \vdots \\ \frac{[\mathrm{q}_{\leq i}]_+ + [\sin\theta(\mathrm{d}_i)]_+}{\sqrt{1+\mathrm{q}_{\leq i}^2}} \\ \vdots \end{bmatrix}
\end{aligned} \tag{147}$$

Finally, considering the residual connection, we obtain the following vectors:

$$\begin{aligned}
\mathbf{x}_i^{(6)} &= \mathbf{h}_i^{(5)} + \begin{bmatrix} \vdots \\ \frac{[\mathrm{q}_{\leq i}]_+ + [\sin\theta(\mathrm{d}_i)]_+}{\sqrt{1+\mathrm{q}_{\leq i}^2}} \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \vdots \\ \cos\phi(i) \\ \sin\phi(i) \\ \vdots \\ \cos\theta(\mathrm{d}_i) \\ \sin\theta(\mathrm{d}_i) \\ \vdots \\ \cos\theta(\mathrm{d}_i+1) \\ \sin\theta(\mathrm{d}_i+1) \\ \tilde{\mathbf{t}}_i \\ \mathrm{q}_i \\ \mathrm{q}_{\leq i} \\ \frac{[\mathrm{q}_{\leq i}]_+ + [\sin\theta(\mathrm{d}_i)]_+}{\sqrt{1+\mathrm{q}_{\leq i}^2}} \\ \mathbf{0} \end{bmatrix}.
\end{aligned} \tag{148}$$

### G.6 Classifier

Finally, the classifier head can determine whether the input sequence belongs to the Dyck language based on the value calculated in the fifth layer; that is, $\frac{[\mathrm{q}_{\leq i}]_+ + [\sin\theta(\mathrm{d}_i)]_+}{\sqrt{1+\mathrm{q}_{\leq i}^2}}$ is non-negative and is zero if and only if the input sequence belongs to the Dyck language. The lower bound of the value when the input does not belong to the $\mathtt{Dyck}_k$ language is calculated as follows:

$$\begin{aligned}
&\frac{[\mathrm{q}_{\leq i}]_+ + [\sin\theta(\mathrm{d}_i)]_+}{\sqrt{1+\mathrm{q}_{\leq i}^2}} \\
&\begin{cases} \geq \frac{1}{\sqrt{1+1^2}} & \text{if } \mathrm{q}_{\leq i} > 0 \\ \geq \frac{\sin\theta(1)}{\sqrt{1+1^2}} & \text{if } \mathrm{d}_i > 0 \\ 0 & \text{otherwise} \end{cases} \\
&\begin{cases} \geq \frac{\sin\theta(1)}{\sqrt{2}} & \text{if } \mathrm{q}_{\leq i} > 0 \vee \mathrm{d}_i > 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{149}$$

Therefore, by subtracting a positive value less than this value as a bias, $\mathrm{sgn}(\cdot)$ can correctly classify whether the sequence belongs to $\mathtt{Dyck}_k$.

For instance, We omit the unnecessary dimensions of input vector $\mathbf{x}_i^{(6)}$ in this layer as follows:

$$\mathbf{x}_i^{(6)} = \begin{bmatrix} \vdots \\ \frac{[\mathrm{q}_{\leq i}]_+ + [\sin\theta(\mathrm{d}_i)]_+}{\sqrt{1+\mathrm{q}_{\leq i}^2}} \\ \vdots \end{bmatrix}. \tag{150}$$

Then, by setting

$$\mathbf{w}^{\mathrm{cls}\top} = \begin{bmatrix} \cdots & -1 & \cdots \end{bmatrix}, \tag{151}$$

$$b^{\mathrm{cls}} = \frac{\sin\theta(1)}{2\sqrt{2}}, \tag{152}$$

we obtain

$$\begin{aligned} &\mathbf{w}^{\mathrm{cls}\top}\mathbf{x}_i^{(6)} + b^{\mathrm{cls}} \\ &= -\frac{[\mathrm{q}_{\leq i}]_+ + [\sin\theta(\mathrm{d}_i)]_+}{\sqrt{1+\mathrm{q}_{\leq i}^2}} + \frac{\sin\theta(1)}{2\sqrt{2}} \\ &\begin{cases} = \frac{\sin\theta(1)}{2\sqrt{2}} & \text{if } w_{0:i} \in \mathtt{Dyck}_k \\ \leq -\frac{\sin\theta(1)}{2\sqrt{2}} & \text{if } w_{0:i} \notin \mathtt{Dyck}_k \end{cases}. \end{aligned} \tag{153}$$

□

# H Proof of Theorem 2

We restate Theorem 2 for convenience.

**Theorem 4** (Restatement of Theorem 2, Transformers with a starting token, $\mathtt{Dyck}_k$ generation)**.** *For all $k$, there exists a 3-layer $O(\log k)$-width causal Transformer network without positional encoding that generates the $\mathtt{Dyck}_k$ language. Each layer incorporates both the residual connection and the layer normalization. This network is followed by a fully-connected layer and softmax layer to output the probability distribution.*

*Proof.* Here, we present a method to construct a Transformer that realizes the $\mathtt{Dyck}_k$ language generation process $p_{\mathtt{Dyck}_k}(\cdot; q, r, \boldsymbol{\pi})$. We assume that the output probabilities take the following form:

$$\begin{bmatrix} p_{\langle_1} \\ \vdots \\ p_{\langle_k} \\ p_{\rangle_1} \\ \vdots \\ p_{\rangle_k} \\ p_{\texttt{<bos>}} \\ p_{\texttt{<eos>}} \end{bmatrix}. \tag{154}$$

As shown in the proof sketch of Theorem 2, each layer performs the following operations. Figure 2 provides the high-level understanding of this Transformer.

**First layer** creates pseudo positional encoding $\begin{bmatrix}\cos\phi(i) & \sin\phi(i)\end{bmatrix}^\top$.

**Second layer** counts depth $\mathrm{d}(w_{0:i})$.

**Third layer** fetches a valid closed bracket if one exists; otherwise, a zero vector is fetched. This operation is achieved by placing attention on the largest position among $\{0\} \cup \{j \mid \mathrm{d}(w_{0:j}) = \mathrm{d}(w_{0:i})\}$.

## H.1 First and second layer

We use the first two layers to create pseudo positional encoding $\begin{bmatrix}\cos\phi(i) & \sin\phi(i)\end{bmatrix}^\top$ and depth $\begin{bmatrix}\cos\theta(\mathrm{d}_i) & \sin\theta(\mathrm{d}_i))\end{bmatrix}^\top$, following the same procedure as described in Appendix G.1 and G.2.

Therefore, the output from the second layer is as follows:

$$\mathbf{x}_i^{(3)} = \begin{bmatrix} \vdots \\ \cos\phi(i) \\ \sin\phi(i) \\ \vdots \\ \cos\theta(\mathrm{d}_i) \\ \sin\theta(\mathrm{d}_i) \\ \vdots \end{bmatrix} \tag{155}$$

## H.2 Third layer

Moreover, in the third layer, we leverage the attention layer to fetch the nearest open bracket with the same depth as the query, in almost the same manner as described in Appendix G.4. The difference from the construction in the previous section is that we replace the query depth $\mathrm{d}_i + 1$ with $\mathrm{d}_i$. Therefore, the output from the attention layer is as follows:

$$\begin{bmatrix} \vdots \\ \cos\phi(i) \\ \sin\phi(i) \\ \vdots \\ \cos\theta(\mathrm{d}_i) \\ \sin\theta(\mathrm{d}_i) \\ \tilde{\mathbf{t}}_i \\ \vdots \end{bmatrix} \tag{156}$$

Then, in the feed-forward network layer, set the parameters as follows:

$$W_1^{(3)} = \begin{bmatrix} \cdots & 0 & 0 & \cdots & 0 & 1 & \mathbf{0}^\top & \cdots \\ \cdots & 0 & 0 & \cdots & 0 & 1 & \mathbf{0}^\top & \cdots \\ \cdots & 0 & 0 & \cdots & 0 & -1 & \mathbf{0}^\top & \cdots \\ \cdots & 0 & 0 & \cdots & 0 & -1 & \mathbf{0}^\top & \cdots \\ \cdots & 0 & 0 & \cdots & 2 & 0 & \mathbf{0}^\top & \cdots \\ & \vdots & \vdots & & \vdots & \vdots & \vdots & \end{bmatrix}, \tag{157}$$

$$W_2^{(3)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix}, \tag{158}$$

$$\boldsymbol{\beta}^{(3)} = \begin{bmatrix} 0 \\ -\epsilon^{(3)} \\ 0 \\ \epsilon^{(3)} \\ 0 \\ \mathbf{0} \end{bmatrix}, \tag{159}$$

$$\boldsymbol{\gamma}^{(3)} = \sqrt{\frac{4}{d_{\text{model}}}}\mathbf{1}, \tag{160}$$

where $\epsilon^{(3)}$ is a positive constant. Then

$$\begin{aligned} &\text{LN}_{\text{RMS}}\left(W_1^{(3)}\mathbf{h}_i^{(3)}\right) \\ &= \boldsymbol{\gamma}^{(3)} \odot \frac{1}{\text{RMS}\left(W_1^{(3)}\mathbf{h}_i^{(3)}\right)} W_1^{(3)}\mathbf{h}_i^{(3)} + \boldsymbol{\beta}^{(3)} \\ &= \sqrt{\frac{4}{d_{\text{model}}}} \frac{1}{\sqrt{\frac{4}{d_{\text{model}}}}} W_1^{(3)}\mathbf{h}_i^{(3)} + \boldsymbol{\beta}^{(3)} \\ &= \begin{bmatrix} \sin\theta(\text{d}_i) \\ \sin\theta(\text{d}_i) \\ -\sin\theta(\text{d}_i) \\ -\sin\theta(\text{d}_i) \\ 2\cos\theta(\text{d}_i) \\ \vdots \end{bmatrix} + \begin{bmatrix} 0 \\ -\epsilon^{(3)} \\ 0 \\ \epsilon^{(3)} \\ 0 \\ \mathbf{0} \end{bmatrix} \\ &= \begin{bmatrix} \sin\theta(\text{d}_i) \\ \sin\theta(\text{d}_i) - \epsilon^{(3)} \\ -\sin\theta(\text{d}_i) \\ -\sin\theta(\text{d}_i) + \epsilon^{(3)} \\ 2\cos\theta(\text{d}_i) \\ \vdots \end{bmatrix}. \end{aligned} \tag{161}$$

because

$$\begin{aligned} &\text{RMS}\left(W_1^{(3)}\mathbf{h}_i^{(3)}\right) \\ &= \sqrt{\frac{4\sin^2\theta(\text{d}_i) + 4\cos^2\theta(\text{d}_i)}{d_{\text{model}}}} \\ &= \sqrt{\frac{4}{d_{\text{model}}}}. \end{aligned} \tag{162}$$

Therefore, we obtain

$$\begin{aligned} &W_2^{(3)}\left[\text{LN}_{\text{RMS}}\left(W_1^{(3)}\mathbf{h}_i^{(3)}\right)\right]_+ \\ &= \begin{bmatrix} \vdots \\ [\sin\theta(\text{d}_i)]_+ \\ [\sin\theta(\text{d}_i) - \epsilon^{(3)}]_+ \\ [-\sin\theta(\text{d}_i)]_+ \\ [-(\sin\theta(\text{d}_i) - \epsilon^{(3)})]_+ \\ \vdots \end{bmatrix} \end{aligned} \tag{163}$$

Finally, we obtain the input vector to the subsequent generator head as follows:

$$\mathbf{x}_i^{(4)} = \begin{bmatrix} \vdots \\ 1 \\ \vdots \\ \tilde{\mathbf{t}}_i \\ [\sin\theta(\text{d}_i)]_+ \\ \left[\sin\theta(\text{d}_i) - \epsilon^{(3)}\right]_+ \\ [-\sin\theta(\text{d}_i)]_+ \\ \left[-(\sin\theta(\text{d}_i) - \epsilon^{(3)})\right]_+ \\ \vdots \end{bmatrix} \tag{164}$$

### H.3 Generator head

For clarity, we implement $W^{\text{gen}}$ as a composition of two linear transformations $W_1^{\text{gen}} \in \mathbb{R}^{(k+4)\times d_{\text{model}}}, W_2^{\text{gen}} \in \mathbb{R}^{(2k+2)\times(k+4)}$ as follows (the transposed matrices are described to accommodate the limited space):

$$W_1^{\text{gen}\top} = \begin{bmatrix} \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \lceil\log_2 k\rceil & \cdots & \lceil\log_2 k\rceil & 0 & 0 & 0 & 0 \\ \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ -\mathbf{t}_{(1)} & \cdots & -\mathbf{t}_{(k)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & \cdots & 0 & 1 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 1 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 1 \\ \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}, \tag{165}$$

$$W_2^{\mathrm{gen}\top} = \begin{bmatrix} O & -C_0^{\mathrm{gen}} I & \mathbf{0} & \mathbf{0} \\ \mathbf{0}^\top & \frac{C_1^{\mathrm{gen}}}{\epsilon^{(3)}}\mathbf{1}^\top & 0 & 0 \\ \mathbf{0}^\top & \frac{C_1^{\mathrm{gen}}}{\epsilon^{(3)}}\mathbf{1}^\top & 0 & 0 \\ \mathbf{0}^\top & \mathbf{0}^\top & 0 & \frac{-C_2^{\mathrm{gen}}}{\epsilon^{(3)}} \\ \mathbf{0}^\top & \mathbf{0}^\top & 0 & \frac{-C_2^{\mathrm{gen}}}{\epsilon^{(3)}} \end{bmatrix}, \tag{166}$$

($k$ dim., $k$ dim., 2 dim.)

$$\mathbf{b}^{\mathrm{gen}} = \begin{bmatrix} C_0^{\mathrm{gen}} + \log \pi_1 \\ \vdots \\ C_0^{\mathrm{gen}} + \log \pi_k \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} \Big\} k \text{ dim.} \\ \\ \\ \Big\} k \text{ dim.} \\ \\ \\ \Big\} 2 \text{ dim.} \end{matrix}, \tag{167}$$

where $C_0^{\mathrm{gen}}$ is a positive constant and

$$C_1^{\mathrm{gen}} = \log\left(\frac{1-q}{q}\right) + C_0^{\mathrm{gen}}, \tag{168}$$

$$C_2^{\mathrm{gen}} = \log\left(\frac{1-r}{r}\right) + C_0^{\mathrm{gen}}. \tag{169}$$

Then, given a sufficiently small constant $\epsilon^{(3)}$,

$$\begin{aligned} & W^{\mathrm{gen}}\mathbf{x}_i^{(4)} + \mathbf{b}^{\mathrm{gen}} \\ &= W_2^{\mathrm{gen}} W_1^{\mathrm{gen}} \mathbf{x}_i^{(4)} + \mathbf{b}^{\mathrm{gen}} \\ &= W_2^{\mathrm{gen}} \begin{bmatrix} -\mathbf{t}_{(1)}^\top \tilde{\mathbf{t}} + \lceil \log_2 k \rceil \\ \vdots \\ -\mathbf{t}_{(k)}^\top \tilde{\mathbf{t}} + \lceil \log_2 k \rceil \\ [\sin\theta(\mathrm{d}_i)]_+ \\ \left[\sin\theta(\mathrm{d}_i) - \epsilon^{(3)}\right]_+ \\ [-\sin\theta(\mathrm{d}_i)]_+ \\ \left[-(\sin\theta(\mathrm{d}_i) - \epsilon^{(3)})\right]_+ \end{bmatrix} + \mathbf{b}^{\mathrm{gen}} \\ &= \begin{bmatrix} C_0^{\mathrm{gen}} + \log \pi_1 \\ \vdots \\ C_0^{\mathrm{gen}} + \log \pi_k \\ \left( -C_0^{\mathrm{gen}} \left( \lceil \log_2 k \rceil - \mathbf{t}_{(1)}^\top \tilde{\mathbf{t}} \right) \right. \\ \left. + C_1^{\mathrm{gen}} \mathbb{I}\left[\mathrm{d}_i \geq 1\right] \right) \\ \vdots \\ \left( -C_0^{\mathrm{gen}} \left( \lceil \log_2 k \rceil - \mathbf{t}_{(k)}^\top \tilde{\mathbf{t}} \right) \right. \\ \left. + C_1^{\mathrm{gen}} \mathbb{I}\left[\mathrm{d}_i \geq 1\right] \right) \\ 0 \\ C_2^{\mathrm{gen}} \mathbb{I}\left[\mathrm{d}_i \leq 0\right] \end{bmatrix}, \end{aligned} \tag{170}$$

where

$$\begin{aligned} & \mathbb{I}\left[\mathrm{d}_i \geq 1\right] \\ &= \frac{[\sin\theta(\mathrm{d}_i)]_+ - \left[\sin\theta(\mathrm{d}_i) - \epsilon^{(3)}\right]_+}{\epsilon^{(3)}} \\ &= \begin{cases} 1 & \text{if } \mathrm{d}_i \geq 1 \\ 0 & \text{otherwise} \end{cases}, \end{aligned} \tag{171}$$

$$\begin{aligned} & \mathbb{I}\left[\mathrm{d}_i \leq 0\right] \\ &= \frac{\left[-(\sin\theta(\mathrm{d}_i) - \epsilon^{(3)})\right]_+ - [-\sin\theta(\mathrm{d}_i)]_+}{\epsilon^{(3)}} \\ &= \begin{cases} 1 & \text{if } \mathrm{d}_i \leq 0 \\ 0 & \text{otherwise} \end{cases}. \end{aligned} \tag{172}$$

### H.4 Softmax

We compute the logit vector separately for the cases where (i) $\mathrm{d}_i = 0$ and (ii) $\mathrm{d}_i \geq 1$. Let $\mathrm{logit}$ be the output logit vector. Note that we identify logit vectors that become identical through translation because they are projected to the same probability vector by the softmax operation. Let $\equiv$ be the equivalence relation on logits.

**(i) In the case of $\mathrm{d}_i = 0$.**

$$\begin{aligned} & \mathrm{logit} \\ &= \begin{bmatrix} C_0^{\mathrm{gen}} + \log \pi_1 \\ \vdots \\ C_0^{\mathrm{gen}} + \log \pi_k \\ \left( -C_0^{\mathrm{gen}} \left( \lceil \log_2 k \rceil - \mathbf{t}_{(1)}^\top \tilde{\mathbf{t}} \right) \right. \\ \left. + C_1^{\mathrm{gen}} \mathbb{I}\left[\mathrm{d}_i \geq 1\right] \right) \\ \vdots \\ \left( -C_0^{\mathrm{gen}} \left( \lceil \log_2 k \rceil - \mathbf{t}_{(k)}^\top \tilde{\mathbf{t}} \right) \right. \\ \left. + C_1^{\mathrm{gen}} \mathbb{I}\left[\mathrm{d}_i \geq 1\right] \right) \\ 0 \\ C_2^{\mathrm{gen}} \mathbb{I}\left[\mathrm{d}_i \leq 0\right] \end{bmatrix} \\ &\equiv \begin{bmatrix} \log r\pi_1 \\ \vdots \\ \log r\pi_k \\ -C_0^{\mathrm{gen}} \left( \lceil \log_2 k \rceil - \mathbf{t}_{(1)}^\top \tilde{\mathbf{t}} + 1 \right) + \log r \\ \vdots \\ -C_0^{\mathrm{gen}} \left( \lceil \log_2 k \rceil - \mathbf{t}_{(k)}^\top \tilde{\mathbf{t}} + 1 \right) + \log r \\ -C_0^{\mathrm{gen}} + \log r \\ \log(1-r) \end{bmatrix} \\ &=: \mathrm{logit}' \end{aligned} \tag{173}$$

To establish the upper bound of the total variation distance, we first derive an upper bound and lower bound for the softmax denominator:

$$\begin{aligned}
&\sum_{l=1}^{K} \exp(\text{logit}'_l) \\
&= \sum_{t=1}^{k} r\pi_t \\
&\quad + \sum_{t=1}^{k} r \exp\left(-C_0^{\text{gen}}\left(\lceil \log_2 k \rceil - \mathbf{t}_{(t)}^{\top}\bar{\mathbf{t}} + 1\right)\right) \\
&\quad + r\exp(-C_0^{\text{gen}}) + 1 - r \\
&\le r + k\exp(-C_0^{\text{gen}}) + \exp(-C_0^{\text{gen}}) + 1 - r \\
&= 1 + (k+1)\exp(-C_0^{\text{gen}}),
\end{aligned} \tag{174}$$

$$\begin{aligned}
&\sum_{l=1}^{K} \exp(\text{logit}'_l) \\
&\ge \sum_{t=1}^{k} r\pi_t + k \cdot 0 + 0 + 1 - r = 1
\end{aligned} \tag{175}$$

Therefore, the lower bound of the total variation distance from the true probability distribution is given by:

$$\begin{aligned}
&2 \cdot \text{TV}(\mathbb{S}(\text{logit}), p_{\texttt{Dyck}_k}(q, r, \boldsymbol{\pi})) \\
&= 2 \cdot \text{TV}(\mathbb{S}(\text{logit}'), p_{\texttt{Dyck}_k}(q, r, \boldsymbol{\pi})) \\
&\le \sum_{t=1}^{k} \frac{(k+1)\exp(-C_0^{\text{gen}})}{1 + (k+1)\exp(-C_0^{\text{gen}})} r\pi_t \\
&\quad + \sum_{t=1}^{k} \exp(-C_0^{\text{gen}}) \\
&\quad + \exp(-C_0^{\text{gen}}) \\
&\quad + \frac{(k+1)\exp(-C_0^{\text{gen}})}{1 + (k+1)\exp(-C_0^{\text{gen}})}(1 - r) \\
&= \frac{(k+1)\exp(-C_0^{\text{gen}})}{1 + (k+1)\exp(-C_0^{\text{gen}})} \\
&\quad + (k+1)\exp(-C_0^{\text{gen}}) \\
&\le 2(k+1)\exp(-C_0^{\text{gen}}).
\end{aligned} \tag{176}$$

**(ii) In the case of $\mathrm{d}_i \ge 1$.** Similar to the case (i), the upper bound of TV distance can be calculated as follows:

$$\begin{aligned}
&\text{TV}(\mathbb{S}(\text{logit}), p_{\texttt{Dyck}_k}(q, r, \boldsymbol{\pi})) \\
&\quad \le (k+1)\exp(-C_0^{\text{gen}}).
\end{aligned} \tag{177}$$

Therefore, for any $\epsilon > 0$, by choosing a constant $C_0^{\text{gen}}$ to satisfy

$$(k+1)\exp(-C_0^{\text{gen}}) < \epsilon \tag{178}$$

$$\Leftrightarrow C_0^{\text{gen}} > \log\frac{k+1}{\epsilon}, \tag{179}$$

then

$$\text{TV}(\mathbb{S}([\text{logit}]), p_{\texttt{Dyck}_k}(q, r, \boldsymbol{\pi})) < \epsilon \tag{180}$$

is satisfied.

Based on the above, the Transformer realizes the $\texttt{Dyck}_k$ language generation process $p_{\texttt{Dyck}_k}(\cdot; q, r, \boldsymbol{\pi})$.

□

# I Proof of Proposition 3

In this section, we show that Transformers are capable of recognizing the Shuffle-Dyck language efficiently with respect to network width, even without the need for a specific positional encoding.

**Proposition 9** (Restatement of Proposition 3, Transformers with a starting token, $\texttt{Shuffle-Dyck}_k$ recognition)**.** *For all $k$, there exists a 3-layer $O(\log k)$-width causal Transformer without positional encoding that recognizes the $\texttt{Shuffle-Dyck}_k$ language. Each layer incorporates both the residual connection and the layer normalization. This network is followed by a fully-connected layer and a sign function to output an acceptance signal.*

*Proof.* We assume the same vector representation as defined in Appendix D:

$$\mathbf{x}_i = \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{d_{\text{model}}}. \tag{181}$$

The first layer creates pseudo positional encoding, and the second layer calculates the depth in almost the same manner as in Theorem 1. However, in the second layer, the feed-forward network layer calculates $[\sin\theta(\mathrm{d}(w_{0:i}))]_+$, instead of calculating $\cos\theta(\mathrm{d}(w_{0:i}))$ and $\sin\theta(\mathrm{d}(w_{0:i}))$; that is, the output from the second layer becomes:

$$\mathbf{x}_i^{(3)} = \begin{bmatrix} \mathbf{t}_i \\ o_i \\ s_i \\ 1 \\ \cos\phi(i) \\ \sin\phi(i) \\ [\sin\theta(\mathrm{d}(w_{0:i}))]_+ \\ \mathbf{0} \end{bmatrix} \tag{182}$$

### I.1 Third layer

The third layer calculates the depth of the substring that matches the same type as the query; that is, this layer computes

$$[\sin(-\theta(\mathrm{d}(w_{0:i} \mid \mathbf{t}_i)))]_+ , \tag{183}$$

where $\mathrm{d}(w_{0:i} \mid \mathbf{t}_i)$ represents the depth when focusing on the substring corresponding to type $\mathbf{t}_i$. For instance, for the input sequence "`([{}])`", this layer outputs the depth vectors corresponding to $[1, 1, 2, 1, 0, 1, 0, 0]$. This is realized in a similar way to Appendix G.2 with slight modification. Specifically, by replacing the query and key matrix with the matrices as follows:

$$W_Q^{(3)} = \begin{bmatrix} C^{(3)}I & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{0}^\top & 0 & 0 & C'^{(3)} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}, \tag{184}$$

$$W_K^{(3)} = \begin{bmatrix} I & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{0}^\top & 0 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}, \tag{185}$$

where $C^{(3)}$ is a positive constant and $C'^{(3)} = C^{(3)}\lceil \log_2 k \rceil + a$.

Then, we obtain

$$\begin{aligned} &\left\langle W_K^{(3)} \mathbf{x}_{i_k}^{(3)}, W_Q^{(3)} \mathbf{x}_{i_q}^{(3)} \right\rangle \\ &= C^{(3)} \langle \mathbf{t}_{i_q}, \mathbf{t}_{i_k} \rangle + C'^{(3)} s_{i_k} \\ &\begin{cases} = C^{(3)} \lceil \log_2 k \rceil & \text{if } \mathbf{t}_{i_k} = \mathbf{t}_{i_q} \\ = C^{(3)} \lceil \log_2 k \rceil + a & \text{if } w_{i_k} = \texttt{<bos>} \\ \leq C^{(3)} (\lceil \log_2 k \rceil - 2) & \text{otherwise} \end{cases} \end{aligned} \tag{186}$$

for $i_q \geq 1$.

Therefore, for a sufficiently large constant $C^{(3)}$, we obtain

$$\begin{aligned} \mathbf{x}_i^{(4)} &= \mathbf{h}_i^{(3)} + \begin{bmatrix} \vdots \\ [\sin(-\theta(\mathrm{d}(w_{0:i} \mid \mathbf{t}_i)))]_+ \\ \vdots \end{bmatrix} \\ &= \begin{bmatrix} \vdots \\ [\sin\theta(\mathrm{d}(w_{0:i}))]_+ \\ \vdots \\ [\sin(-\theta(\mathrm{d}(w_{0:i} \mid \mathbf{t}_i)))]_+ \\ \vdots \end{bmatrix}. \end{aligned} \tag{187}$$

### I.2 Fourth layer

The fourth layer computes a necessary and sufficient condition for the string $w_{0:n}$ to belong to $\texttt{Shuffle-Dyck}_k$, which is described in the following lemma.

**Lemma 1.** *The simultaneous satisfaction of the following two conditions constitutes a necessary and sufficient condition for $w_{0:n}$ to belong to $\texttt{Shuffle-Dyck}_k$:*

**Condition (i)** $\mathrm{d}(w_{0:n}) \leq 0$.

**Condition (ii)** $\forall i \in [n].\ \mathrm{d}(w_{0:i} \mid \mathbf{t}_i) \geq 0$ .

*Proof.* It is sufficient to show that

$$\begin{aligned} &(\mathrm{d}(w_{0:n}) = 0) \wedge (\forall i \in [n].\ \mathrm{d}(w_{0:i} \mid \mathbf{t}_i) \geq 0) \\ &\iff w_{0:n} \in \texttt{Shuffle-Dyck}_k, \end{aligned} \tag{188}$$

because

$$\begin{aligned} &(\mathrm{d}(w_{0:n}) \leq 0) \wedge (\forall i \in [n].\ \mathrm{d}(w_{0:i} \mid \mathbf{t}_i) \geq 0) \\ &\implies (\mathrm{d}(w_{0:n}) \leq 0) \wedge (\mathrm{d}(w_{0:n}) \geq 0) \\ &\implies \mathrm{d}(w_{0:n}) = 0. \end{aligned} \tag{189}$$

**Proof of necessity.** Since it is evident that

$$\begin{aligned} &w_{0:n} \in \texttt{Shuffle-Dyck}_k \\ &\implies \mathrm{d}(w_{0:n}) = 0, \end{aligned} \tag{190}$$

it is sufficient to show that

$$\begin{aligned} &w_{0:n} \in \texttt{Shuffle-Dyck}_k \\ &\implies \forall i \in [n].\ \mathrm{d}(w_{0:i} \mid \mathbf{t}_i) \geq 0. \end{aligned} \tag{191}$$

Here, when $\exists i \in [n].\ \mathrm{d}(w_{0:i} \mid \mathbf{t}_i) < 0$ is satisfied, the substring of type-$\mathbf{t}_i$ is not correctly nested, indicating that $\forall i \in [n].\ \mathrm{d}(w_{0:i} \mid \mathbf{t}_i) \geq 0$ for $w_{0:n} \in \texttt{Shuffle-Dyck}_k$.

**Proof of sufficiency.** For any $t \in [k]$, there is no $i \in [n]$ such that $\mathrm{d}(w_{0:i} \mid t) < 0$, indicating that for any $t \in [k]$, the substring of type-$t$ is a correct prefix for $\texttt{Dyck}_1$. In addition, if there exists $t \in [k]$ such that $\mathrm{d}(w_{0:n} \mid t) > 0$, since $\mathrm{d}(w_{0:n}) = 0$, there exists $t \in [k]$ such that $\mathrm{d}(w_{0:n} \mid t) < 0$. This contradicts to the fact that the substring of type-$t$ is a correct prefix for $\texttt{Dyck}_1$, indicating that for any $t \in [k]$, $\mathrm{d}(w_{0:n} \mid t) = 0$ holds.

□

To check whether the two conditions specified in Lemma 1, it is sufficient to calculate

$$\begin{aligned}&\mathrm{q}_{\texttt{Shuffle-Dyck}}(w_{0:i})\\&= [\sin\theta(\mathrm{d}(w_{0:i}))]_+\\&\quad + \frac{1}{i+1}\sum_{j=0}^{i}[\sin\theta(-\mathrm{d}(w_{0:j} \mid \mathbf{t}_j))]_+,\end{aligned} \tag{192}$$

because $\mathrm{q}_{\texttt{Shuffle-Dyck}}(w_{0:i})$ is always non-negative and becomes 0 if and only if the two conditions above are simultaneously satisfied.

We then show how to implement a Transformer block that computes $\mathrm{q}_{\texttt{Shuffle-Dyck}}(w_{0:i})$ in the fourth layer.

**Fourth layer — Attention layer**

We omit the unnecessary dimensions of input vector $\mathbf{x}_i^{(4)}$ in this layer as follows:

$$\mathbf{x}_i^{(4)} = \begin{bmatrix}\vdots\\ [\sin(-\theta(\mathrm{d}(w_{0:i} \mid \mathbf{t}_i)))]_+\\ \vdots\end{bmatrix}. \tag{193}$$

By setting the parameters $W_Q^{(4)} = O, W_K^{(4)} = O, W_V^{(4)} = \begin{bmatrix}\vdots\\ \cdots\ 1\ \cdots\\ \vdots\end{bmatrix}$, we obtain the mean vector $\frac{1}{i+1}\sum_{j=0}^{i}[\sin\theta(-\mathrm{d}(w_{0:j} \mid \mathbf{t}_j))]_+$. Therefore, by adding the mean vector to the dimension corresponding to $[\sin(\theta(\mathrm{d}(w_{0:i})))]_+$, we obtain

$$\mathbf{h}_i^{(4)} = \begin{bmatrix}\vdots\\ \mathrm{q}_{\texttt{Shuffle-Dyck}}(w_{0:i})\\ \vdots\end{bmatrix}. \tag{194}$$

**Fourth layer — Feed-forward network layer**

The feed-forward network has nothing to do. By setting $W_1^{(4)} = O, W_2^{(4)} = O$, we obtain $\mathbf{x}_i^{(5)} = \mathbf{h}_i^{(4)}$.

### I.3 Classifier

The classifier head can simply determine the string as positive if $\mathrm{q}_{\texttt{Shuffle-Dyck}}(w_{0:i})$ is 0 and as negative if it is strictly greater than 0.

Specifically, we omit the unnecessary dimensions of input vector $\mathbf{x}_i^{(5)}$ in this layer as follows:

$$\mathbf{x}_i^{(5)} = \begin{bmatrix}\vdots\\ \mathrm{q}_{\texttt{Shuffle-Dyck}}(w_{0:i})\\ \vdots\end{bmatrix}. \tag{195}$$

Set the parameter $\mathbf{w}^{\mathrm{cls}} \in \mathbb{R}^{d_{\mathrm{model}}}$ and $b^{\mathrm{cls}} \in \mathbb{R}$ as follows:

$$\mathbf{w}^{\mathrm{cls}} = \begin{bmatrix}\vdots\\ -1\\ \vdots\end{bmatrix}, \tag{196}$$

$$b^{\mathrm{cls}} = \epsilon. \tag{197}$$

Then, the desired computation can be achieved because

$$\mathbf{w}^{\mathrm{cls}\top}\mathbf{x}_i^{(5)} + b^{\mathrm{cls}} \begin{cases} = \epsilon & \text{if } w_{1:i} \in \texttt{Shuffle-Dyck}_k\\ < 0 & \text{if } w_{1:i} \notin \texttt{Shuffle-Dyck}_k\end{cases}. \tag{198}$$

□

## J Proof of Proposition 4

**Proposition 10** (Restatement of Proposition 4, Transformers with a starting token, $\texttt{Shuffle-Dyck}_k$ generation)**.** *For all $k$, there exists a 3-layer $O(k)$-width causal Transformer without positional encoding that generates the $\texttt{Shuffle-Dyck}_k$ language. Each layer incorporates both the residual connection and the layer normalization. This network is followed by a fully-connected layer and softmax layer to output the probability distribution.*

*Proof.* Here, unlike the other sections, we assume one-hot vectors as the bracket-type embeddings, where we multiply by $+1$ for open brackets and by $-1$ for closed brackets. For instance, "$\langle_2$" is mapped into $\begin{bmatrix}0 & 1 & \cdots & 0\end{bmatrix}^\top$ and "$\rangle_1$" is mapped into $\begin{bmatrix}-1 & 0 & \cdots & 0\end{bmatrix}^\top$. In addition, we prepare an $O(k)$-dimensional zero vector that acts as a memory. Therefore, the input vector without positional encoding $\mathbf{x}_i^{(1)}$ becomes as follows:

$$\mathbf{x}_i^{(1)} = \begin{bmatrix}\mathbf{t}_i\\ \mathbf{0}\end{bmatrix}\begin{matrix}\}k \text{ dim.}\\ \}(d_{\mathrm{model}} - k) \text{ dim.}\end{matrix} \in \mathbb{R}^{d_{\mathrm{model}}}. \tag{199}$$

### J.1 First layer

**First layer — Attention layer**

In the first layer, using uniform attention, the query at position $i$ computes the mean vector of $\{\mathbf{t}_j\}_{j=0}^i$; that is, the output of the attention layer becomes:

$$\mathbf{h}_i^{(1)} = \begin{bmatrix}\mathbf{t}_i\\ \frac{1}{i+1}\sum_{j=0}^{i}\mathbf{t}_j\\ \mathbf{0}\end{bmatrix}. \tag{200}$$

**First layer — Feed-forward network layer**

We omit the unnecessary dimensions of input vector $\mathbf{h}_i^{(1)}$ in this layer as follows:

$$\mathbf{h}_i^{(1)} = \begin{bmatrix} \mathbf{t}_i \\ \frac{1}{i+1}\sum_{j=0}^{i}\mathbf{t}_j \\ \vdots \end{bmatrix}. \tag{201}$$

Set the parameters $W_1^{(1)}, W_2^{(1)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ and $\boldsymbol{\beta}^{(1)}, \boldsymbol{\gamma}^{(1)} \in \mathbb{R}^{d_{\text{model}}}$ as follows:

$$W_1^{(1)} = \begin{bmatrix} O & -I & \cdots \\ O & -I & \cdots \\ I & O & \cdots \\ \vdots & \vdots & \end{bmatrix}, \tag{202}$$

$$W_2^{(1)} = \begin{bmatrix} \vdots & \vdots & \vdots & \\ -I & I & O & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}, \tag{203}$$

$$\boldsymbol{\beta}^{(1)} = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \\ \mathbf{0} \\ \vdots \end{bmatrix}, \tag{204}$$

$$\boldsymbol{\gamma}^{(1)} = \frac{1}{\epsilon\sqrt{d_{\text{model}}}}\mathbf{1}. \tag{205}$$

Then, we obtain

$$\begin{aligned}
&W_2^{(1)}\left[\mathrm{LN}_{\mathrm{RMS}}\left(W_1^{(1)}\mathbf{h}_i^{(1)}\right)\right]_+ \\
&= W_2^{(1)}\left[\mathrm{LN}_{\mathrm{RMS}}\left(\begin{bmatrix} -\frac{1}{i+1}\sum_{j=0}^{i}\mathbf{t}_j \\ -\frac{1}{i+1}\sum_{j=0}^{i}\mathbf{t}_j \\ \mathbf{t}_i \\ \vdots \end{bmatrix}\right)\right]_+ \\
&= W_2^{(1)}\begin{bmatrix} -\frac{\sum_{j=0}^{i}\mathbf{t}_j}{\epsilon\sqrt{(i+1)^2+2\|\sum_{j=0}^{i}\mathbf{t}_j\|_2^2}} \\ \mathbf{1} - \frac{\sum_{j=0}^{i}\mathbf{t}_j}{\epsilon\sqrt{(i+1)^2+2\|\sum_{j=0}^{i}\mathbf{t}_j\|_2^2}} \\ \frac{(i+1)\mathbf{t}_i}{\epsilon\sqrt{(i+1)^2+2\|\sum_{j=0}^{i}\mathbf{t}_j\|_2^2}} \\ \vdots \end{bmatrix}_+ \\
&= \begin{bmatrix} \vdots \\ \mathbb{I}[\mathrm{d}(w_{0:i} \mid t=1) \le 0] \\ \vdots \\ \mathbb{I}[\mathrm{d}(w_{0:i} \mid t=k) \le 0] \\ \vdots \end{bmatrix},
\end{aligned} \tag{206}$$

where

$$\begin{aligned}
&\mathbb{I}[\mathrm{d}(w_{0:i} \mid t=t') \le 0] \\
&= -\left[-\frac{\sum_{j=0}^{i}t_{j,t'}}{\epsilon\sqrt{(i+1)^2+2\|\sum_{j=0}^{i}\mathbf{t}_j\|_2^2}}\right]_+ \\
&\quad + \left[1 - \frac{\sum_{j=0}^{i}t_{j,t'}}{\epsilon\sqrt{(i+1)^2+2\|\sum_{j=0}^{i}\mathbf{t}_j\|_2^2}}\right]_+ \\
&= \begin{cases} 1 & \text{if } \mathrm{d}(w_{0:i} \mid t=t') \le 0 \\ 0 & \text{otherwise} \end{cases}.
\end{aligned} \tag{207}$$

Finally, considering the residual connection, we obtain

$$\begin{aligned}
\mathbf{x}_i^{(2)} &= \mathbf{h}_i^{(1)} + \begin{bmatrix} \vdots \\ \mathbb{I}[\mathrm{d}(w_{0:i} \mid t=1) \le 0] \\ \vdots \\ \mathbb{I}[\mathrm{d}(w_{0:i} \mid t=k) \le 0] \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{t}_i \\ \frac{1}{i+1}\sum_{j=0}^{i}\mathbf{t}_j \\ \mathbb{I}[\mathrm{d}(w_{0:i} \mid t=1) \le 0] \\ \vdots \\ \mathbb{I}[\mathrm{d}(w_{0:i} \mid t=k) \le 0] \\ \vdots \end{bmatrix}.
\end{aligned} \tag{208}$$

### J.2 Generator head

We omit the unnecessary dimensions of input vector $\mathbf{x}_i^{(2)}$ in this layer as follows:

$$\mathbf{x}_i^{(2)} = \begin{bmatrix} \vdots \\ \mathbf{m}(w_{0:i}) \\ \vdots \end{bmatrix}, \tag{209}$$

where

$$\mathbf{m}(w_{0:i}) = \begin{bmatrix} \mathbb{I}[\mathrm{d}(w_{0:i} \mid t=1) \le 0] \\ \vdots \\ \mathbb{I}[\mathrm{d}(w_{0:i} \mid t=k) \le 0] \end{bmatrix}. \tag{210}$$

Set the parameters $W^{\text{gen}} \in \mathbb{R}^{(2k+2)\times d_{\text{model}}}, \mathbf{b}^{\text{gen}} \in \mathbb{R}^{(2k+2)}$ as follows:

$$W^{\text{gen}} = \begin{bmatrix} \cdots & O & \cdots \\ \cdots & -C^{\text{gen}}I & \cdots \\ \cdots & \mathbf{0}^\top & \cdots \\ \cdots & C^{\text{gen}}\mathbf{1}^\top & \cdots \end{bmatrix} \begin{matrix} \}k \text{ dim.} \\ \}k \text{ dim.} \\ \}1 \text{ dim.,} \\ \}1 \text{ dim.} \end{matrix} \tag{211}$$

$$\mathbf{b}^{\text{gen}} = \begin{bmatrix} \log \boldsymbol{\pi} \\ \log\left(\frac{1-q}{q}\right)\overline{\boldsymbol{\pi}} \\ -C^{\text{gen}} \\ \log\left(\frac{1-r}{r}\right) - kC^{\text{gen}} \end{bmatrix} \begin{matrix} \}k \text{ dim.} \\ \}k \text{ dim.} \\ \}1 \text{ dim.} \\ \}1 \text{ dim.} \end{matrix}. \quad (212)$$

Then, we obtain

$$\begin{aligned} &\text{logit} \\ &= W^{\text{gen}}\mathbf{x}_i^{(2)} + \mathbf{b}^{\text{gen}} \\ &= \begin{bmatrix} \log \boldsymbol{\pi} \\ \log\left(\frac{1-q}{q}\right)\overline{\boldsymbol{\pi}} - C^{\text{gen}}\mathbf{m}(w_{0:i}) \\ -C^{\text{gen}} \\ \log\left(\frac{1-r}{r}\right) - C^{\text{gen}}(k - \mathbf{1}^\top \mathbf{m}(w_{0:i})) \end{bmatrix}. \end{aligned} \quad (213)$$

**Softmax**

Similar to the Appendix H, it is possible to show that the $\texttt{Shuffle-Dyck}_k$ language generation process can be approximated with arbitrary precision. Here, for clarity, we treat $-C^{\text{gen}}$ as a masking operation in the softmax function and show how it realizes the language generation process.

**(i) in case that $\forall t'.\, \mathrm{d}(w_{0:i} \mid t = t') = 0$.** Since $\mathbf{m}(w_{0:i}) = \mathbf{1}$,

$$\text{logit} = \begin{bmatrix} \log \boldsymbol{\pi} \\ \log\left(\frac{1-q}{q}\right)\overline{\boldsymbol{\pi}} - C^{\text{gen}}\mathbf{1} \\ -C^{\text{gen}} \\ \log\left(\frac{1-r}{r}\right) \end{bmatrix}. \quad (214)$$

Therefore,

$$\begin{aligned} \mathbb{S}(\text{logit}) &\simeq \mathbb{S}\left(\begin{bmatrix} \log \boldsymbol{\pi} \\ \texttt{masked} \\ \texttt{masked} \\ \log\left(\frac{1-r}{r}\right) \end{bmatrix}\right) \\ &= \mathbb{S}\left(\begin{bmatrix} \log r\boldsymbol{\pi} \\ \texttt{masked} \\ \texttt{masked} \\ \log(1-r) \end{bmatrix}\right), \end{aligned} \quad (215)$$

indicating that the $\texttt{Shuffle-Dyck}_k$ language generation process is realized.

**(i) in case that $\exists t'.\, \mathrm{d}(w_{0:i} \mid t = t') > 0$.** Since $\mathbf{1}^\top \mathbf{m}(w_{0:i}) \leq k - 1$,

$$\begin{aligned} &\text{logit} \\ &= \begin{bmatrix} \log \boldsymbol{\pi} \\ \log\left(\frac{1-q}{q}\right)\overline{\boldsymbol{\pi}} - C^{\text{gen}}\mathbf{m}(w_{0:i}) \\ -C^{\text{gen}} \\ \log\left(\frac{1-r}{r}\right) - C^{\text{gen}}(k - \mathbf{1}^\top \mathbf{m}(w_{0:i})) \end{bmatrix}. \end{aligned} \quad (216)$$

Therefore, we obtain

$$\begin{aligned} &\mathbb{S}(\text{logit}) \\ &\simeq \mathbb{S}\left(\begin{bmatrix} \log \boldsymbol{\pi} \\ \log\left(\frac{1-q}{q}\right)\overline{\boldsymbol{\pi}} - C^{\text{gen}}\mathbf{m}(w_{0:i}) \\ \texttt{masked} \\ \texttt{masked} \end{bmatrix}\right) \\ &= \mathbb{S}\left(\begin{bmatrix} \log q\boldsymbol{\pi} \\ \log(1-q)\overline{\boldsymbol{\pi}} - C^{\text{gen}}\mathbf{m}(w_{0:i}) \\ \texttt{masked} \\ \texttt{masked} \end{bmatrix}\right). \end{aligned} \quad (217)$$

In addition, the $t$-th element of $\log \overline{\boldsymbol{\pi}} - C^{\text{gen}}\mathbf{m}(w_{0:i})$ is masked if and only if the depth of type $t$ is 0 or less than 0, indicating that the $\texttt{Shuffle-Dyck}_k$ language generation process is also realized in this case.

□

## K Proof of Proposition 5

**Proposition 11** (Restatement of Proposition 5)**.** *There is no network whose width grows strictly slower than $k/\log k$ that generates $\texttt{Shuffle-Dyck}_k$; that is, if*

$$\lim_{k\to\infty} \frac{d_{\text{model}}(k)}{k/\log k} = 0 \quad (218)$$

*holds, then there exists $k_0$ such that for any $k \geq k_0$, networks with $d_{\text{model}}(k)$-width cannot generate $\texttt{Shuffle-Dyck}_k$.*

We provide the proof sketch first. Then, we show some lemmas in Section K.1 and give a proof of Proposition 5 in Section K.2.

*Proof sketch.* We give a proof by contradiction. Consider the $2^k$ different input strings: concerning the $l \in [2^k]$-th input, when the $t$-th bit of the binary representation of $l$ is 1, we add an open bracket of type $t$. For example, when $k = 2$, we consider the following $2^2$ inputs:

$$\begin{aligned} 00 &\mapsto \texttt{<bos>}, \\ 01 &\mapsto \texttt{<bos>}\langle_1, \\ 10 &\mapsto \texttt{<bos>}\langle_2, \\ 11 &\mapsto \texttt{<bos>}\langle_1\langle_2. \end{aligned} \quad (219)$$

Then, it is necessary to satisfy the following $2^k$ constraints to generate $\texttt{Shuffle-Dyck}_k$ correctly: concerning the $l$-th constraint, if the $t_0$ and $t_1$-th bit of the binary representation of $l$ are 0 and 1, respectively, the $t_1$-th logit is strictly greater than

the $t_0$-th logit. However, there is no linear transformation $\mathbb{R}^{d_{\text{model}}} \to \mathbb{R}^k$ that satisfies the constraints above. □

In this section, we explicitly express the dependence of $d_{\text{model}}$ on $k$, denoting it by $d(k)$ for clarity. Additionally, we occasionally use the concise notation $e$ instead of $\exp$ to conserve space.

### K.1 Preliminary Lemmas

**Definition 11** (Subspace)**.** *Given a vector set $\{\mathbf{w}_t\}_{t=1}^k \subset \mathbb{R}^{d(k)}$. Let $\text{bin}(l) = l_1 \cdots l_k$ be the binary representation of an integer $l \in [2^k]$ and $\text{bin}_t(l) = l_t$ be the $t$-th bit of $\text{bin}(l)$. Then, we define subspace $R(l)(\subset \mathbb{R}^{d(k)})$ as follows:*

$$R(l) = \left\{ \mathbf{x} \middle| \begin{matrix} (\mathbf{w}_{t_1} - \mathbf{w}_{t_0})^\top \mathbf{x} > 0 \\ \forall t_0 \in \text{id}_0(l), t_1 \in \text{id}_1(l) \end{matrix} \right\}, \quad (220)$$

*where*

$$\text{id}_0(l) = \{t \mid \text{bin}_t(l) = 0\}, \quad (221)$$
$$\text{id}_1(l) = \{t \mid \text{bin}_t(l) = 1\}. \quad (222)$$

*In addition, we say $R(l)$ and $R(l')$ are distinct if $R(l) \cap R(l') = \emptyset$. Moreover, we say the subspace set $\{R(\cdot)\}$ is distinct if for any two subspaces are distinct.*

Intuitively, for $\mathbf{x} \in R(l)$, $\mathbf{w}_{t_1}^\top \mathbf{x} > \mathbf{w}_{t_0}^\top \mathbf{x}$ holds, indicating the logit for type-$t_1$ is greater than that for type-$t_0$.

**Lemma 2.** *$R(l)$ and $R(l')$ are distinct if there exists $t \neq t'$ such that $t \in \text{id}_0(l) \wedge t' \in \text{id}_1(l) \wedge t \in \text{id}_1(l') \wedge t' \in \text{id}_0(l')$.*

*Proof.* Since $t \in \text{id}_0(l) \wedge t' \in \text{id}_1(l)$,

$$R(l) \subset \left\{ \mathbf{x} \middle| (\mathbf{w}_{t'} - \mathbf{w}_t)^\top \mathbf{x} > 0 \right\} \quad (223)$$

holds. In contrast, since $t \in \text{id}_1(l') \wedge t' \in \text{id}_0(l')$,

$$R(l') \subset \left\{ \mathbf{x} \middle| (\mathbf{w}_t - \mathbf{w}_{t'})^\top \mathbf{x} > 0 \right\} \quad (224)$$

holds, indicating $R(l) \cap R(l') = \emptyset$. □

**Lemma 3.** *$R(l)$ and $R(l')$ are distinct if $l \neq l' \wedge \#_1(l) = \#_1(l')$ holds, where $\#_1(l)$ is the number of ones in $\text{bin}(l)$.*

*Proof.* Since $l \neq l'$, there exists a digit $t$ such that $l_t \neq l'_t$. Without loss of generality, we can assume that $l_t = 1 \wedge l'_t = 0$. In addition, since $l$ and $l'$ have same number of ones, there exists $t'$ such that $l_{t'} = 0 \wedge l'_{t'} = 1$, indicating that $R(l)$ and $R(l')$ are distinct from Lemma 2. □

**Lemma 4.** *For any integer $m \geq 1$,*

$$e^{\frac{11}{12}} m^{m+\frac{1}{2}} e^{-m} \leq m! \leq e^{\frac{13}{12}} m^{m+\frac{1}{2}} e^{-m} \quad (225)$$

*holds.*

*Proof.* From the results in Robbins (1955),

$$\frac{m! e^m}{m^{m+\frac{1}{2}}} = C e^{r_m}, \quad (226)$$

holds for $m \geq 1$, where $C \in \left(e^{\frac{11}{12}}, e^{\frac{12}{13}}\right)$ and $r_m \in \left(\frac{1}{12m+1}, \frac{1}{12m}\right)$. Therefore

$$\begin{aligned} \frac{m! e^m}{m^{m+\frac{1}{2}}} &\in \left(e^{\left(\frac{11}{12} + \frac{1}{12m+1}\right)}, e^{\left(\frac{12}{13} + \frac{1}{12m}\right)}\right) \\ &\subseteq \left(e^{\frac{11}{12}}, e^{\frac{13}{12}}\right), \end{aligned} \quad (227)$$

indicating that the inequality holds for $m \geq 1$. □

**Lemma 5.** *For $k > 2$ and an integer set $\left[2^k\right] = \{0, \cdots, 2^k - 1\}$. Then, at least $\left\lfloor \sqrt{2}^k \right\rfloor$-size distinct subspace set is necessary to make $R(l) \subset \mathbb{R}^{d_{\text{model}}}$ non-empty for any $l \in \left[2^k\right]$.*

*Proof.* In case that $k$ is an even number,

$$\left| \left\{ l \middle| \#_1(l) = \frac{k}{2} \right\} \right| = \binom{k}{k/2}, \quad (228)$$

On the other hand, in case that $k$ is an odd number,

$$\left| \left\{ l \middle| \#_1(l) = \frac{k-1}{2} \right\} \right| = \binom{k}{(k-1)/2}. \quad (229)$$

When $k$ is even, at least $\binom{k}{k/2}$-size distinct subspace set is necessary because from Lemma 3, $\{R(l)\}_{l \in \left\{l' \middle| \#_1(l') = \frac{k}{2}\right\}}$ is distinct. Here,

$$\begin{aligned} &\binom{k}{k/2} \\ &= \frac{k!}{(k/2)!(k/2)!} \\ &\geq \frac{e^{\frac{11}{12}} k^{k+\frac{1}{2}} e^{-k}}{\left(e^{\frac{13}{12}} \left(\frac{k}{2}\right)^{\frac{k}{2}+\frac{1}{2}} e^{-\frac{k}{2}}\right)^2} \text{ (Lemma 4)} \\ &= 2e^{-\frac{37}{144}} \frac{2^k}{\sqrt{k}} \\ &\geq \sqrt{2}^k \frac{\sqrt{2}^k}{\sqrt{k}} \geq \sqrt{2}^k \end{aligned} \quad (230)$$

holds, indicating that $\left\lfloor \sqrt{2}^k \right\rfloor$-size distinct subspace set is necessary. Similarly, when $k$ is odd,

$$\begin{aligned} &\binom{k}{(k-1)/2} \\ &\geq \sqrt{2}^k \frac{\sqrt{2}^k}{(k+1)/\sqrt{k}} \geq \sqrt{2}^k \end{aligned} \tag{231}$$

holds, leading to the same result. □

**Lemma 6.** *For any real number $x > 1$,*

$$x^{\frac{1}{x}} < 1 + \frac{\log x + 1}{x} \tag{232}$$

*holds.*

*Proof.*

$$\begin{aligned} x^{\frac{1}{x}} &= \exp\left(\frac{1}{x}\log x\right) \\ &= \sum_{p=0}^{\infty} \frac{\left(\frac{1}{x}\log x\right)^p}{p!} \\ &= 1 + \frac{1}{x}\log x + \sum_{p=2}^{\infty} \frac{\left(\frac{1}{x}\log x\right)^p}{p!} \\ &\leq 1 + \frac{1}{x}\log x + \left(\frac{1}{x}\log x\right)^2 \cdot \sum_{p=2}^{\infty} \frac{1}{p!} \\ &= 1 + \frac{1}{x}\log x + \frac{1}{x}\cdot\frac{(\log x)^2}{x} \cdot \left(\sum_{p=0}^{\infty}\frac{1}{p!} - 2\right) \\ &< 1 + \frac{1}{x}\log x + \frac{1}{x}\cdot 1 \cdot (e-2) \\ &< 1 + \frac{\log x + 1}{x}. \end{aligned} \tag{233}$$

□

We cite Lemma 7 stated in Bagdasaryan (2023). Note that we modify the statement to align with this paper.

**Lemma 7** (Bagdasaryan (2023))**.** *Let $G(d(k), m)$ be the maximum number of regions that are separated by $m$ hyperplanes in $\mathbb{R}^{d(k)}$. Then,*

$$G(d(k), m) = \sum_{d=0}^{d(k)} \binom{m}{d}, \tag{234}$$

*where*

$$\binom{m}{d} = \begin{cases} \frac{m!}{d!(m-d)!} & \text{if } d \leq m \\ 0 & \text{if } d > m \end{cases}. \tag{235}$$

**Lemma 8.** *For any function $d(k) : \mathbb{N} \to \mathbb{N}$ that satisfies $d(k) = o(k/\log k)$, $\log G\left(d(k), \binom{k}{2}\right)$ is a sub-linear function; that is, when*

$$\lim_{k\to\infty} \frac{d(k)}{k/\log k} = 0 \tag{236}$$

*holds, the following equation holds:*

$$\frac{\log G\left(d(k), \binom{k}{2}\right)}{k} \underset{k\to\infty}{\longrightarrow} 0. \tag{237}$$

*Proof.* Since $d(k)$ grows strictly slower than $k/\log k$, there exists $k_0$ such that for any $k > k_0$, $d(k) < \frac{k^2}{2}$ holds. We assume $k > k_0$ for the remainder.

$$\begin{aligned} &G\left(d(k), \binom{k}{2}\right) \\ &= \sum_{d=0}^{d(k)} \binom{\binom{k}{2}}{d} \text{ (from Lemma 7)} \\ &\leq \sum_{d=0}^{d(k)} \binom{k^2}{d} \\ &\leq \sum_{d=0}^{d(k)} \binom{k^2}{d(k)} \left(\text{because } d(k) < \frac{k^2}{2}\right) \\ &\leq 2d(k)\frac{k^2(k^2-1)\cdots(k^2-d(k)+1)}{d(k)!} \\ &\leq \frac{2d(k)k^{2d(k)}}{e^{\frac{11}{12}}d(k)^{d(k)+\frac{1}{2}}e^{-d(k)}} \text{ (from Lemma 4)} \\ &\leq 2\sqrt{d(k)}e^{d(k)}\left(\frac{k^2}{d(k)}\right)^{\frac{d(k)}{k^2}k^2} \\ &\leq 2\sqrt{d(k)}e^{d(k)}\left(1 + \frac{\log\left(\frac{k^2}{d(k)}\right)+1}{\frac{k^2}{d(k)}}\right)^{k^2} \\ &\qquad \text{(from Lemma 6)} \\ &\leq 2\sqrt{d(k)}e^{d(k)}\left(1 + \frac{\tilde{d}(k)}{k^2}\right)^{\frac{k^2}{\tilde{d}(k)}\tilde{d}(k)} \\ &< 2\sqrt{d(k)}e^{d(k)+\tilde{d}(k)}, \end{aligned} \tag{238}$$

where

$$\tilde{d}(k) = d(k)\left(\log\left(\frac{k^2}{d(k)}\right) + 1\right). \tag{239}$$

Therefore,

$$\begin{aligned}
&\frac{\log G\left(d(k), \binom{k}{2}\right)}{k} \\
&\leq \frac{\log\left(2\sqrt{d(k)}e^{d(k)+\tilde{d}(k)}\right)}{k} \\
&\leq \frac{\log 2\sqrt{d(k)} + \tilde{d}(k) + \tilde{d}(k)}{k} \\
&= \frac{\log 2\sqrt{d(k)} + 2d(k)\left(\log\left(\frac{k^2}{d(k)}\right)+1\right)}{k} \\
&\leq \frac{\log 2\sqrt{d(k)} + d(k)\cdot 6\log k}{k} \\
&\leq \frac{\log 2\sqrt{d(k)}}{k} + 6\frac{d(k)}{k/\log k} \\
&\underset{k\to\infty}{\longrightarrow} 0,
\end{aligned} \tag{240}$$

indicating $\log G\left(d(k), \binom{k}{2}\right)$ is a sub-linear function. □

**Lemma 9.** *For any function $d(k) : \mathbb{N} \to \mathbb{N}$ that satisfies $d(k) = o(k/\log k)$ and $\nu > 0$,*

$$\lim_{k\to\infty} \frac{G\left(d(k), \binom{k}{2}\right)}{\exp(\nu k)} = 0 \tag{241}$$

*holds.*

*Proof.* From Lemma 8, since $d(k)$ grows strictly slower than $k/\log k$, $\log G\left(d(k), \binom{k}{2}\right)$ grows sub-linearly. Therefore, for any $\nu > 0$, there exists $k_0$ such that for any $k > k_0$,

$$\frac{\log G\left(d(k), \binom{k}{2}\right)}{k} < \frac{\nu}{2} \tag{242}$$

holds; thus, for any $\nu$ and $k > k_0$,

$$\begin{aligned}
&\lim_{k\to\infty} \frac{G\left(d(k), \binom{k}{2}\right)}{\exp(\nu k)} \\
&= \lim_{k\to\infty} \exp\left(\log G\left(d(k), \binom{k}{2}\right) - \nu k\right) \\
&= \lim_{k\to\infty} \exp\left(\left(\frac{\log G\left(d(k), \binom{k}{2}\right)}{k} - \nu\right) k\right) \\
&\leq \lim_{k\to\infty} \exp\left(-\frac{\nu}{2}k\right) \\
&= 0.
\end{aligned} \tag{243}$$

□

### K.2 Main proof

*Proof.* We derive a contradiction by assuming the existence of a $d(k)$-width network and a generator head $f^{\text{gen}} : \mathbb{R}^{d(k)} \to \mathbb{R}^{2k+2}$ that generates $\texttt{Shuffle-Dyck}_k$. Denote the matrix of the generator head by

$$W^{\text{gen}} = \begin{bmatrix} \vdots \\ \mathbf{w}_1^\top \\ \vdots \\ \mathbf{w}_k^\top \\ \vdots \end{bmatrix} \begin{matrix} \} k \text{ dim.} \\ \\ \} k \text{ dim.} \\ \\ \} 2 \text{ dim.} \end{matrix} \in \mathbb{R}^{(2k+2)\times d(k)}. \tag{244}$$

Take into account the $2^k$ vectors $\{\mathbf{x}_l\}_{l\in[2^k]}$ corresponding to the input strings described in the proof sketch; that is, $\text{bin}_t(l) = 0$ means the type $t$ is closed, while $\text{bin}_t(l) = 1$ means the type $t$ is unclosed. Here, the generator head satisfies

$$\mathbf{w}_{t_0}^\top \mathbf{x}_l > \mathbf{w}_{t_1}^\top \mathbf{x}_l, \tag{245}$$

for any $l \in [2^k]$ and for any $t_1 \in \text{id}_1(l), t_0 \in \text{id}_0(l)$. This is because the logit for the unclosed type must be greater than that for the closed type.

Consider the subspace set defined by $\{\mathbf{w}_t\}_{t=1}^k$, since $\mathbf{x}_l \in R(l)$, from Lemma 5, there exists at least $\left\lfloor \sqrt{2}^k \right\rfloor$-size distinct subspace set.

However, the generator head can create at most $G\left(d(k), \binom{k}{2}\right)$-size separated regions in $\mathbb{R}^{d(k)}$, leading a contradiction: the number of separable regions increases strictly slower than the necessary size of distinct subspace set from Lemma 9. □

## L Proof of Proposition 6

**Proposition 12** (Restatement of Proposition 6)**.** *Assume that there exists a linear mapping such that the transformed embeddings are distinct from each other and have a constant 2-norm. Then, there exists a Transformer block without a starting token that creates a pseudo starting signal $\hat{s}_i$ for any string $w_{1:n}$ whose first two tokens are different, where*

$$\hat{s}_i = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{otherwise} \end{cases}. \tag{246}$$

*Specifically, this block transforms the constants-padded vector $\hat{\mathbf{x}}_i$ as follows:*

$$\hat{\mathbf{x}}_i = \begin{bmatrix} \mathbf{x}_i \\ \vdots \\ 0 \end{bmatrix} \mapsto \begin{bmatrix} \mathbf{x}_i \\ \vdots \\ \hat{s}_i \end{bmatrix}. \tag{247}$$

*Proof.* Assume the extended input representation $\hat{\mathbf{x}}_i \in \mathbb{R}^{d'_{\text{model}}}$, where $d'_{\text{model}} = 2d_{\text{model}} + 2$, instead of the original representation $\mathbf{x}_i \in \mathbb{R}^{d_{\text{model}}}$ as follows:

$$\hat{\mathbf{x}}_i = \begin{bmatrix} \mathbf{x}_i \\ \mathbf{0} \\ 1 \\ 0 \end{bmatrix} \begin{matrix} \}d_{\text{model}} \text{ dim.} \\ \}d_{\text{model}} \text{ dim.} \\ \}1 \text{ dim.} \\ \}1 \text{ dim.} \end{matrix}. \tag{248}$$

The attention layer, leveraging the uniform attention, transforms the vector into

$$\begin{bmatrix} \mathbf{x}_i \\ \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j \\ 1 \\ 0 \end{bmatrix}. \tag{249}$$

Then, in the feed-forward network layer, the first linear transformation calculates $\mathbf{x}_i - \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j$. The 2-norm of this vector is 0 if and only if $\mathbf{x}_i = \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j$. Thanks to the subsequent layer normalization, the larger the 2-norm of the vector $\mathbf{x}_i - \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j$ becomes, the smaller the transformed value of the constant 1 becomes. This allows the subsequent ReLU activations and the linear transformation to implement the conditional branch. We then show the specific implementation.

Set the parameters $W_1, W_2 \in \mathbb{R}^{d'_{\text{model}} \times d'_{\text{model}}}$ and $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^{d'_{\text{model}}}$ as follows:

$$W_1 = \begin{bmatrix} I & -I & \mathbf{0} & \mathbf{0} \\ O & O & \mathbf{0} & \mathbf{0} \\ \mathbf{0}^\top & \mathbf{0}^\top & 1 & 0 \\ \mathbf{0}^\top & \mathbf{0}^\top & 0 & 0 \end{bmatrix}, \tag{250}$$

$$W_2 = \begin{bmatrix} O & O & \mathbf{0} & \mathbf{0} \\ O & O & \mathbf{0} & \mathbf{0} \\ \mathbf{0}^\top & \mathbf{0}^\top & 0 & 0 \\ \mathbf{0}^\top & \mathbf{0}^\top & \frac{1}{\epsilon} & 0 \end{bmatrix}, \tag{251}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ -1+\epsilon \\ 0 \end{bmatrix}, \tag{252}$$

$$\boldsymbol{\gamma} = \sqrt{\frac{1}{d'_{\text{model}}}}\mathbf{1}, \tag{253}$$

where $\epsilon$ is a positive constant.

Then, the output of the RMS layer normalization is given by

$$\begin{aligned} &\text{LN}_{\text{RMS}}\left(W_1 \mathbf{h}_i\right) \\ &= \text{LN}_{\text{RMS}}\left(\begin{bmatrix} \mathbf{x}_i - \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j \\ \mathbf{0} \\ 1 \\ 0 \end{bmatrix}\right) \\ &= \frac{1}{\|W_1 \mathbf{h}_i\|_2} \begin{bmatrix} \mathbf{x}_i - \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j \\ \mathbf{0} \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ -1+\epsilon \\ 0 \end{bmatrix} \\ &= \frac{1}{\|W_1 \mathbf{h}_i\|_2} \begin{bmatrix} \mathbf{x}_i - \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j \\ \mathbf{0} \\ 1 - \|W_1 \mathbf{h}_i\|_2 (1-\epsilon) \\ 0 \end{bmatrix}. \end{aligned} \tag{254}$$

Therefore, the output of the feed-forward network layer is given by

$$\begin{aligned} &W_2 \left[ \frac{1}{\|W_1 \mathbf{h}_i\|_2} \begin{bmatrix} \mathbf{x}_i - \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j \\ \mathbf{0} \\ 1 - \|W_1 \mathbf{h}_i\|_2 (1-\epsilon) \\ 0 \end{bmatrix} \right]_+ \\ &= \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ 0 \\ \frac{1}{\epsilon}\left[\frac{1}{\|W_1 \mathbf{h}_i\|_2} - 1 + \epsilon\right]_+ \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ 0 \\ \mathbb{I}\left[\mathbf{x}_i = \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j\right] \end{bmatrix} \end{aligned} \tag{255}$$

The reason why the last equality holds is explained below: since

$$\begin{aligned} \|W_1 \mathbf{h}_i\|_2^2 &= \left\| \mathbf{x}_i - \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j \right\|_2^2 + 1^2 \\ &\begin{cases} = 1 & \text{if } \mathbf{x}_i = \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j \\ > 1 & \text{if } \mathbf{x}_i \neq \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j \end{cases}, \end{aligned} \tag{256}$$

the entry 1 is transformed to 1 if $\mathbf{x}_i = \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j$; otherwise, the entry becomes less than 1. Therefore, given a sufficiently small constant $\epsilon$,

$$\begin{aligned} &\frac{1}{\|W_1 \mathbf{h}_i\|_2} - 1 + \epsilon \\ &\begin{cases} = \epsilon & \text{if } \mathbf{x}_i = \frac{1}{i}\sum_{j=1}^{i} \mathbf{x}_j \\ < 0 & \text{otherwise} \end{cases} \end{aligned} \tag{257}$$

holds, indicating

$$\frac{1}{\epsilon}\left[\frac{1}{\|W_1\mathbf{h}_i\|_2}-1+\epsilon\right]_+ = \mathbb{I}\left[\mathbf{x}_i = \frac{1}{i}\sum_{j=1}^{i}\mathbf{x}_j\right]. \tag{258}$$

Finally, we give a proof of the following proposition: $\mathbf{x}_1 \neq \mathbf{x}_2 \Leftrightarrow \mathbf{x}_i \neq \frac{1}{i}\sum_{j=1}^{i}\mathbf{x}_j$ (for $i \geq 2$). Let $C_2$ be the constant 2-norm of the embeddings. Then,

$$\begin{aligned}\mathbf{x}_i = \frac{1}{i}\sum_{j=1}^{i}\mathbf{x}_j &\Longrightarrow \left\langle \mathbf{x}_i, \frac{1}{i}\sum_{j=1}^{i}\mathbf{x}_j \right\rangle = C_2^2 \\ &\Longrightarrow \frac{1}{i}\sum_{j=1}^{i}\langle \mathbf{x}_i, \mathbf{x}_j\rangle = C_2^2 \\ &\Longrightarrow \forall j \in [i].\mathbf{x}_i = \mathbf{x}_j.\end{aligned} \tag{259}$$

This is because

$$\frac{1}{i}\sum_{j=1}^{i}\langle \mathbf{x}_i, \mathbf{x}_j\rangle \leq \frac{1}{i}\sum_{j=1}^{i}\|\mathbf{x}_i\|\|\mathbf{x}_j\| = C_2^2 \tag{260}$$

holds for any $i$, and the equality holds if and only if $\forall j \in [i].\mathbf{x}_i = \mathbf{x}_j$ holds. On the other hand, the converse is straightforward. Therefore,

$$\forall i \geq 2.\mathbf{x} \neq \frac{1}{i}\sum_{j=1}^{i}\mathbf{x}_j \tag{261}$$

$$\Longleftrightarrow \forall i \geq 2.\exists j \in [i].\mathbf{x}_i \neq \mathbf{x}_j \tag{262}$$

$$\Longleftrightarrow \mathbf{x}_1 \neq \mathbf{x}_2, \tag{263}$$

indicating that when $\mathbf{x}_1 \neq \mathbf{x}_2$, A Transformer block can create a pseudo starting signal $\hat{s}_i$ by itself. □

## M Proof of Corollary 1

Here, we present a method to construct a Transformer without positional encoding and <bos> that recognizes the $\mathtt{Dyck}_k$ language for an input string whose first two characters are different.

**Corollary 3** (Restatement of Corollary 1, Transformers without a starting token, $\mathtt{Dyck}_k$ probabilistic recognition)**.** *Assume the same assumption as in Proposition 6. There exists a* 9*-layer causal Transformer without a starting token that recognizes the* $\mathtt{Dyck}_k$ *language with probability at least* $1 - 1/k$.

*Proof.* Here, for clarity, we omit the specific implementation except that of the fourth layer. Instead, we outline the construction.

Firstly, using the starting token created by Proposition 6, create pseudo positional encoding, which allows the Transformer to compute the same representation $\mathbf{x}_{i-1}$ as used in Theorem 1. The proof of Theorem 1 does not require the query to assign an attention score on itself; thus, it is possible to calculate $\mathrm{q}(w_{0:i})$ by making the query matrix focus on $\mathbf{x}_i^{(\ell)}$ and the key/value matrices focus on $\mathbf{x}_{i-1}^{(\ell)}$. Moreover, by focusing solely on $\mathbf{x}_{i-1}^{(\ell)}$, it is possible to compute $\bigwedge_{j=0}^{i-1} Q(w_{0:j-1})$ in the same manner as described in Appendix G. Finally, to check whether the input string is a prefix for $\mathtt{Dyck}_k$, it is sufficient to compute $Q(w_{0:i}) \wedge \bigwedge_{j=0}^{i-1} Q(w_{0:j-1})$.

Specifically, the following nine layers can recognize the $\mathtt{Dyck}_k$ language.

**First layer** creates a pseudo starting signal $\hat{s}_i$ using Proposition 6.

**Second and third layers** create vectors corresponding $\phi(i-1)$ and $\phi(i)$, respectively.

**Fourth layer** computes the same representation $\mathbf{x}_{i-1}$ as in Appendix D.

**Fifth and sixth layers** create vectors corresponding $\mathrm{d}(w_{0:i-1})$ and $\mathrm{d}(w_{0:i}) + 1$, respectively.

**Seventh and eighth layers** compute $\mathrm{q}_{i-1}$ and $\mathrm{q}_i$, which correspond to the propositional variables $Q(w_{0:i-1})$ and $Q(w_{0:i})$, respectively.

**Ninth layer** computes $Q(w_{0:i}) \wedge \bigwedge_{j=0}^{i-1} Q(w_{0:j}) \wedge \mathrm{d}(w_{0:i}) + 1 = 1$.

### M.1 How to compute $\mathbf{x}_{i-1}$

The attention layer in the fourth layer, leveraging the pseudo positional encoding, computes

$$\mathbf{h}_i^{(4)} = \begin{bmatrix} \vdots \\ \hat{s}_i \\ \vdots \\ \mathbf{t}_{i_{\mathrm{prev}}} \\ o_{i_{\mathrm{prev}}} \\ \vdots \end{bmatrix}, \tag{264}$$

where

$$\mathbf{t}_{i_{\mathrm{prev}}} = \begin{cases} \mathbf{t}_1 & \text{if } i = 1 \\ \mathbf{t}_{i-1} & \text{if } i > 1 \end{cases}, \tag{265}$$

$$o_{i_{\mathrm{prev}}} = \begin{cases} \mathbf{o}_1 & \text{if } i = 1 \\ \mathbf{o}_{i-1} & \text{if } i > 1 \end{cases}. \tag{266}$$

In the subsequent feed-forward network layer, set the parameters $W_1^{(4)}, W_2^{(4)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ and $\boldsymbol{\beta}^{(4)}, \boldsymbol{\gamma}^{(4)} \in \mathbb{R}^{d_{\text{model}}}$ as follows:

$$W_1^{(4)} = \begin{bmatrix} \cdots & \mathbf{0} & \cdots & I & \mathbf{0} & \cdots \\ \cdots & \mathbf{0} & \cdots & -I & \mathbf{0} & \cdots \\ \cdots & 0 & \cdots & \mathbf{0}^\top & 1 & \cdots \\ \cdots & 0 & \cdots & \mathbf{0}^\top & -1 & \cdots \\ \cdots & C^{(4)} & \cdots & \mathbf{0}^\top & 0 & \cdots \\ & \vdots & & \vdots & \vdots & \end{bmatrix}, \tag{267}$$

$$W_2^{(4)} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \\ I & -I & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{0}^\top & \mathbf{0}^\top & 1 & -1 & 0 & \cdots \\ \mathbf{0}^\top & \mathbf{0}^\top & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix}, \tag{268}$$

$$\boldsymbol{\beta}^{(4)} = \mathbf{0}, \tag{269}$$

$$\boldsymbol{\gamma}^{(4)} = \frac{1}{\sqrt{d_{\text{model}}}} \begin{bmatrix} \sqrt{2\left(\lceil \log_2 k \rceil + 1\right)}\mathbf{1} \\ \sqrt{2\left(\lceil \log_2 k \rceil + 1\right)}\mathbf{1} \\ \sqrt{2\left(\lceil \log_2 k \rceil + 1\right)} \\ \sqrt{2\left(\lceil \log_2 k \rceil + 1\right)} \\ 1 \\ \vdots \end{bmatrix}. \tag{270}$$

Given a sufficiently large constant $C^{(4)}$, since we obtain

$$\begin{aligned} &\text{LN}_{\text{RMS}}\left(\begin{bmatrix} \mathbf{t}_{i_{\text{prev}}} \\ -\mathbf{t}_{i_{\text{prev}}} \\ o_{i_{\text{prev}}} \\ -o_{i_{\text{prev}}} \\ C^{(4)}\hat{s}_i \\ \vdots \end{bmatrix}\right) \\ &= \begin{cases} \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix} & \text{if } i = 1 \\ \begin{bmatrix} \mathbf{t}_{i_{\text{prev}}} \\ -\mathbf{t}_{i_{\text{prev}}} \\ o_{i_{\text{prev}}} \\ -o_{i_{\text{prev}}} \\ 0 \\ \vdots \end{bmatrix} & \text{otherwise} \end{cases}, \end{aligned} \tag{271}$$

the output of the RMS layer normalization is given by:

$$\text{LN}_{\text{RMS}}\left(W_1^{(4)}\mathbf{h}_i^{(4)}\right) = \begin{bmatrix} \mathbf{t}_{i-1} \\ -\mathbf{t}_{i-1} \\ o_{i-1} \\ -o_{i-1} \\ s_{i-1} \\ \vdots \end{bmatrix}. \tag{272}$$

Here, it should be noted that although the left-hand and the right-hand side of Equation (271) are not exactly the same, they can be regarded equivalent based on a similar discussion to that in Section O.

Therefore, the output of the feed-forward network layer is given by:

$$\begin{aligned} &W_2^{(4)}\left[\text{LN}_{\text{RMS}}\left(W_1^{(4)}\mathbf{h}_i^{(4)}\right)\right]_+ \\ &= W_2^{(4)}\begin{bmatrix} [\mathbf{t}_{i-1}]_+ \\ [-\mathbf{t}_{i-1}]_+ \\ [o_{i-1}]_+ \\ [-o_{i-1}]_+ \\ s_{i-1} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \mathbf{t}_{i-1} \\ o_{i-1} \\ s_{i-1} \\ \vdots \end{bmatrix}, \end{aligned} \tag{273}$$

indicating that $\mathbf{x}_{i-1}$ can be computed correctly.

Finally, since the probability of outputting the same type of open bracket as the first one is $\frac{r}{k}$, the first two characters are different with at least a probability of $1 - \frac{1}{k}$, which completes the proof. □

## N Proof of Corollary 2

Here, we present a method to construct a Transformer without positional encoding and <bos> that realizes the $\texttt{Dyck}_k$ language generation process $p_{\texttt{Dyck}_k}(\cdot; q, r, \boldsymbol{\pi})$.

**Corollary 4** (Restatement of Corollary 2, Transformers without a starting token, $\texttt{Dyck}_k$ subset generation)**.** *Assume the same assumption as in Proposition 6. There exists a 7-layer causal Transformer without a starting token that can generate a subset of* $\texttt{Dyck}_k$ *where the first two characters are different; that is, the Transformer can generate all possible subsequent sequences when there is an input string whose first two characters are different.*

*Proof.* We assume that the output probabilities take the following form, which is the same as Section

H:

$$\begin{bmatrix} p_{\langle_1} \\ \vdots \\ p_{\langle_k} \\ p_{\rangle_1} \\ \vdots \\ p_{\rangle_k} \\ p_{\texttt{<bos>}} \\ p_{\texttt{<eos>}} \end{bmatrix}. \tag{274}$$

We omit the specific implementation. Instead, we outline the construction for clarity as in Appendix M. Similar to the Transformer without <bos> that recognizes the $\texttt{Dyck}_k$ language, the query vector does not need to assign attention to itself; thus, by making the query matrix focus on $\mathbf{x}_i^{(\ell)}$ and the key/value matrices focus on $\mathbf{x}_{i-1}^{(\ell)}$, the desired behavior can be realized.

Specifically, the five layers described below generate the $\texttt{Dyck}_k$ language when the first two characters of the input string are different.

**First layer** creates a pseudo starting signal $\hat{s}_i$ using Proposition 6.

**Second and third layers** create vectors corresponding $\phi(i-1)$ and $\phi(i)$, respectively.

**Fourth layer** computes the same representation $\mathbf{x}_{i-1}$ as in Appendix D, in the same way as in Section M.1.

**Fifth and sixth layers** create vectors corresponding $\mathrm{d}(w_{0:i-1})$ and $\mathrm{d}(w_{0:i})$, respectively.

**Seventh layer** enables each closed bracket to fetch the nearest depth-matched open bracket.

□

# O Validity of Treating Softmax Attention as Hardmax Attention

In our constructive proofs, we occasionally treat softmax attention as hardmax attention. In this section, we validate these theoretical results; that is, we show that if the vector fetched by hardmax attention is included in the finite set of candidates, the subsequent feed-forward network layer can transform the vector obtained by softmax attention into that obtained by hardmax attention when the assigned attention weight exceeds a certain threshold. Here, we discuss the fourth attention layer described in Appendix G.4 as an example.

## O.1 Threshold of attention strength

**Lemma 10.** *Assume a vector set $\{\mathbf{y}_i\}_{i=1}^n \subset \{1, 0, -1\}^d$. Let $\tilde{\mathbf{y}}_{i,\mathrm{H}}$ and $\tilde{\mathbf{y}}_{i,\mathrm{S}}$ be the vectors obtained by hardmax attention and softmax attention among $\{\mathbf{y}_i\}_{i=1}^n$, respectively. Then, regarding softmax attention, if a query assigns the attention greater than $\frac{2}{3}$ on the target token, the vector obtained by hardmax attention $\tilde{\mathbf{y}}_{i,\mathrm{H}}$ can be identified by referencing $\tilde{\mathbf{y}}_{i,\mathrm{S}}$.*

*Proof.* When a query assigns greater than $\frac{2}{3}$ on the target token, there exists $\rho > \frac{2}{3}$ and $\mathbf{y}_{\mathrm{CH}}$ in the convex hull of $\{\mathbf{y}_i\}$ such that

$$\tilde{\mathbf{y}}_{i,\mathrm{S}} = \rho \tilde{\mathbf{y}}_{i,\mathrm{H}} + (1-\rho)\mathbf{y}_{\mathrm{CH}}. \tag{275}$$

Since absolute value of each elements in $\mathbf{y}_{\mathrm{CH}}$ is at most 1, regarding the $l$-th element $(\tilde{\mathbf{y}}_{i,\mathrm{S}})_l$ of $\tilde{\mathbf{y}}_{i,\mathrm{S}}$,

$$\rho(\tilde{\mathbf{y}}_{i,\mathrm{H}})_l - (1-\rho) \le (\tilde{\mathbf{y}}_{i,\mathrm{S}})_l, \tag{276}$$

$$(\tilde{\mathbf{y}}_{i,\mathrm{S}})_l \le \rho(\tilde{\mathbf{y}}_{i,\mathrm{H}})_l + (1-\rho) \tag{277}$$

hold. Here,

$$\begin{cases} \rho \cdot (-1) + (1-\rho) < \rho \cdot 0 - (1-\rho) \\ \rho \cdot 0 + (1-\rho) < \rho \cdot 1 - (1-\rho) \end{cases} \tag{278}$$

$$\iff \rho > \frac{2}{3} \tag{279}$$

is satisfied, indicating that the original values are identifiable. □

## O.2 Recovering the original value with feed-forward network layer

Here, we show how to implement the feed-forward network layer that recovers the vectors obtained by hardmax attention and realizes the computation in the fourth layer. Since this recovery is feasible if the attention weight is greater than $\frac{2}{3}$ from Lemma 10, we set this threshold to $\frac{4}{5}$ as an example.

Intuitively, we implement a function similar to a step function using the ReLU activations to recover vectors that include errors produced by the prior softmax attention. Specifically, since the element of the $\left[\mathbf{t}_i - \tilde{\mathbf{t}}_i\right]_+$ and $o_i + 1$ take values of $0, 1, 2$, we implement the recovering function $\mathrm{Recov}(y)$ as follows:

$$\begin{aligned} &\mathrm{Recov}(y) \\ &= \left[\frac{y}{\epsilon} - \frac{9}{20\epsilon}\right]_+ + \left[\frac{y}{\epsilon} - \left(1 + \frac{9}{20\epsilon}\right)\right]_+ \\ &\quad + \left[\frac{y}{\epsilon} - \frac{19}{15\epsilon}\right]_+ - \left[\frac{y}{\epsilon} - \left(1 + \frac{19}{15\epsilon}\right)\right]_+ \end{aligned} \tag{280}$$

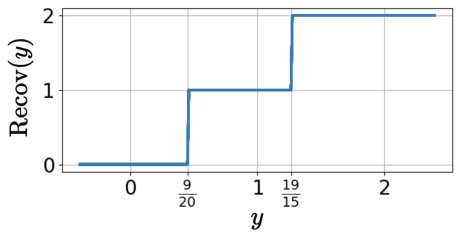


Figure 7: Illustration of the behavior of the recovering function defined in Equation (280).

The behavior of the recovering function is illustrated in Figure 7.

Then, we show the specific implementation that realizes the recovering function. Let $\tilde{\mathbf{t}}'$ be a bracket-type embedding fetched by softmax attention and $\tilde{\mathbf{t}}_i$ be that fetched by hardmax attention. We omit the unnecessary dimensions of input vector $\mathbf{h}_i^{(4)}$ in this layer as follows:

$$\mathbf{h}_i^{(4)} = \begin{bmatrix} \mathbf{t}_i \\ o_i \\ \vdots \\ 1 \\ \vdots \\ \tilde{\mathbf{t}}'_i \\ \vdots \end{bmatrix}. \tag{281}$$

Set the parameter $W_1^{(4)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ and $\boldsymbol{\beta}^{(4)}, \boldsymbol{\gamma}^{(4)} \in \mathbb{R}^{d_{\text{model}}}$ as follows:

$$W_1^{(4)} = \begin{bmatrix} I & \mathbf{0} & \cdots & \mathbf{0} & \cdots & -I & \cdots \\ I & \mathbf{0} & \cdots & \mathbf{0} & \cdots & -I & \cdots \\ I & \mathbf{0} & \cdots & \mathbf{0} & \cdots & -I & \cdots \\ I & \mathbf{0} & \cdots & \mathbf{0} & \cdots & -I & \cdots \\ -I & \mathbf{0} & \cdots & \mathbf{0} & \cdots & I & \cdots \\ -I & \mathbf{0} & \cdots & \mathbf{0} & \cdots & I & \cdots \\ -I & \mathbf{0} & \cdots & \mathbf{0} & \cdots & I & \cdots \\ -I & \mathbf{0} & \cdots & \mathbf{0} & \cdots & I & \cdots \\ \mathbf{0}^\top & 1 & \cdots & 1 & \cdots & \mathbf{0}^\top & \cdots \\ \mathbf{0}^\top & 1 & \cdots & 1 & \cdots & \mathbf{0}^\top & \cdots \\ \mathbf{0}^\top & 1 & \cdots & 1 & \cdots & \mathbf{0}^\top & \cdots \\ \mathbf{0}^\top & 1 & \cdots & 1 & \cdots & \mathbf{0}^\top & \cdots \\ \mathbf{0}^\top & 0 & \cdots & C & \cdots & \mathbf{0}^\top & \cdots \\ \mathbf{0}^\top & 0 & \cdots & C & \cdots & \mathbf{0}^\top & \cdots \\ \vdots & \vdots & & \vdots & & \vdots & \end{bmatrix}, \tag{282}$$

$$\boldsymbol{\beta}^{(4)} = \begin{bmatrix} -\frac{9}{20\epsilon}\mathbf{1} \\ -(1+\frac{9}{20\epsilon})\mathbf{1} \\ -\frac{19}{15\epsilon}\mathbf{1} \\ -(1+\frac{19}{15\epsilon})\mathbf{1} \\ -\frac{9}{20\epsilon}\mathbf{1} \\ -(1+\frac{9}{20\epsilon})\mathbf{1} \\ -\frac{19}{15\epsilon}\mathbf{1} \\ -(1+\frac{19}{15\epsilon})\mathbf{1} \\ -\frac{9}{20\epsilon} \\ -(1+\frac{9}{20\epsilon}) \\ -\frac{19}{15\epsilon} \\ -(1+\frac{19}{15\epsilon}) \\ -\frac{9}{20\epsilon} \\ -(1+\frac{9}{20\epsilon}) \\ \mathbf{0} \end{bmatrix}, \tag{283}$$

$$\boldsymbol{\gamma}^{(4)} = \frac{1}{\epsilon}\sqrt{\frac{2C^2}{d_{\text{model}}}} \begin{bmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ \frac{1}{C} \\ \frac{1}{C} \\ \mathbf{0} \end{bmatrix}, \tag{284}$$

where $\epsilon$ is a positive constant that satisfies $\epsilon < \frac{1}{10}$. Then, we obtain

$$\begin{aligned} &\text{LN}_{\text{RMS}}\left(W_1^{(4)}\mathbf{h}_i^{(4)}\right) \\ &= \text{LN}_{\text{RMS}}\left(\begin{bmatrix} \mathbf{t}_i - \tilde{\mathbf{t}}'_i \\ \mathbf{t}_i - \tilde{\mathbf{t}}'_i \\ \mathbf{t}_i - \tilde{\mathbf{t}}'_i \\ \mathbf{t}_i - \tilde{\mathbf{t}}'_i \\ -(\mathbf{t}_i - \tilde{\mathbf{t}}'_i) \\ -(\mathbf{t}_i - \tilde{\mathbf{t}}'_i) \\ -(\mathbf{t}_i - \tilde{\mathbf{t}}'_i) \\ -(\mathbf{t}_i - \tilde{\mathbf{t}}'_i) \\ o_i + 1 \\ o_i + 1 \\ o_i + 1 \\ o_i + 1 \\ C \\ C \\ \vdots \end{bmatrix}\right) \end{aligned} \tag{285}$$

Here,

$$\sup \frac{1}{\mathrm{RMS}\left(W_1^{(4)}\mathbf{h}_i^{(4)}\right)} = \sqrt{\frac{d_{\text{model}}}{2C^2}}, \tag{286}$$

$$\begin{aligned}
&\inf \frac{1}{\mathrm{RMS}\left(W_1^{(4)}\mathbf{h}_i^{(4)}\right)} \\
&= \sqrt{\frac{d_{\text{model}}}{8\cdot 2^2\lceil\log_2 k\rceil + 4\cdot 2^2 + 2\cdot C^2}} \\
&= \sqrt{\frac{d_{\text{model}}}{2C^2}}\left(1+\frac{16\lceil\log_2 k\rceil+8}{C^2}\right)^{-\frac{1}{2}} \\
&\geq \sqrt{\frac{d_{\text{model}}}{2C^2}}\left(1-\delta_C\right),
\end{aligned} \tag{287}$$

where $\delta_C = \frac{8\lceil\log_2 k\rceil+4}{C^2}$.

Moreover, when $\rho = \frac{4}{5}$, since

$$\tilde{t}'_{i,l} \in \begin{cases} \left[\frac{3}{5}, 1\right] & \text{if } \tilde{t}_{i,l} = 1 \\ \left[-\frac{1}{5}, \frac{1}{5}\right] & \text{if } \tilde{t}_{i,l} = 0 \\ \left[-1, -\frac{3}{5}\right] & \text{if } \tilde{t}_{i,l} = -1 \end{cases}, \tag{288}$$

$$t_{i,l} - \tilde{t}'_{i,l} \in \begin{cases} \left[\frac{8}{5}, 2\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = 2 \\ \left[\frac{3}{5}, \frac{6}{5}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = 1 \\ \left[-\frac{2}{5}, \frac{2}{5}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = 0 \\ \left[-\frac{6}{5}, -\frac{3}{5}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = -1 \\ \left[-2, -\frac{8}{5}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = -2 \end{cases}. \tag{289}$$

Therefore,

$$\sqrt{\frac{2C^2}{d_{\text{model}}}} \cdot \frac{t_{i,l} - \tilde{t}'_{i,l}}{\mathrm{RMS}\left(W_1^{(4)}\mathbf{h}_i^{(4)}\right)} \in \begin{cases} \left[\frac{8(1-\delta_C)}{5}, 2\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = 2 \\ \left[\frac{3(1-\delta_C)}{5}, \frac{6}{5}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = 1 \\ \left[-\frac{2}{5}, \frac{2}{5}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = 0 \\ \left[-\frac{6}{5}, -\frac{3(1-\delta_C)}{5}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = -1 \\ \left[-2, -\frac{8(1-\delta_C)}{5}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = -2 \end{cases}. \tag{290}$$

By setting $C$ to satisfy

$$\begin{aligned}
&\begin{cases} \frac{6}{5} < \frac{8(1-\delta_C)}{5} \\ \frac{2}{5} < \frac{3(1-\delta_C)}{5} \end{cases} \\
&\Leftrightarrow \delta_C < \frac{1}{4} \\
&\Leftrightarrow C > 4\sqrt{2\lceil\log_2 k\rceil+1},
\end{aligned} \tag{291}$$

these five intervals become disjoint. We proceed with our discussion under the assumption $C > 2\sqrt{6}\cdot\sqrt{2\lceil\log_2 k\rceil+1} \Leftrightarrow \delta_C < \frac{1}{6}$ as an example. In this case,

$$\sqrt{\frac{2C^2}{d_{\text{model}}}} \cdot \frac{t_{i,l} - \tilde{t}'_{i,l}}{\mathrm{RMS}\left(W_1^{(4)}\mathbf{h}_i^{(4)}\right)} \in \begin{cases} \left[\frac{4}{3}, 2\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = 2 \\ \left[\frac{1}{2}, \frac{6}{5}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = 1 \\ \left[-\frac{2}{5}, \frac{2}{5}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = 0 \\ \left[-\frac{6}{5}, -\frac{1}{2}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = -1 \\ \left[-2, -\frac{4}{3}\right] & \text{if } t_{i,l} - \tilde{t}_{i,l} = -2 \end{cases}. \tag{292}$$

Similarly,

$$\sqrt{\frac{2C^2}{d_{\text{model}}}} \cdot \frac{o_i + 1}{\mathrm{RMS}\left(W_1^{(4)}\mathbf{h}_i^{(4)}\right)} \begin{cases} = 0 & \text{if } o_i + 1 = 0 \\ \in \left[\frac{5}{6}, 1\right] & \text{if } o_i + 1 = 1 \\ \in \left[\frac{5}{3}, 2\right] & \text{if } o_i + 1 = 2 \end{cases}, \tag{293}$$

$$\left(\sqrt{\frac{2C^2}{d_{\text{model}}}} \cdot \frac{1}{C}\right) \cdot \frac{C}{\mathrm{RMS}\left(W_1^{(4)}\mathbf{h}_i^{(4)}\right)} \in \left[\frac{5}{6}, 1\right] \tag{294}$$

hold. Therefore, by implementing the recovering function $\mathrm{Recov}(\cdot)$ defined in Equation (280), the vectors obtained by hardmax attention are recovered. Finally, by setting $W_2^{(4)} \in \mathbb{R}^{d_{\text{model}}\times d_{\text{model}}}$

$$W_2^{(4)} = \begin{bmatrix} \vdots \\ \mathbf{w}_2^{(4)\top} \\ \vdots \end{bmatrix}, \tag{295}$$

where

$$\mathbf{w}_2^{(4)} = \begin{bmatrix} -4 \cdot \mathbf{1} \\ 4 \cdot \mathbf{1} \\ -4 \cdot \mathbf{1} \\ 4 \cdot \mathbf{1} \\ -4 \cdot \mathbf{1} \\ 4 \cdot \mathbf{1} \\ -4 \cdot \mathbf{1} \\ 4 \cdot \mathbf{1} \\ 8(\lceil \log_2 k \rceil + 1) \\ -8(\lceil \log_2 k \rceil + 1) \\ 8(\lceil \log_2 k \rceil + 1) \\ -8(\lceil \log_2 k \rceil + 1) \\ 2 \\ -2 \\ \mathbf{0} \end{bmatrix}, \tag{296}$$

the desired vector is obtained; that is, the feed-forward network layer computes

$$\begin{bmatrix} \vdots \\ \tilde{\mathrm{q}}(w_{0:i}) \\ \vdots \end{bmatrix}, \tag{297}$$

where

$$\begin{aligned} \tilde{\mathrm{q}}(w_{0:i}) = &- 4\|\mathbf{t}_i - \tilde{\mathbf{t}}_i\|_1 \\ &+ 8(\lceil \log_2 k \rceil + 1)(o_i + 1) + 2, \end{aligned} \tag{298}$$

which the same expression as $\mathrm{q}'(w_{0:i})$ in Equation (125); that is, $\tilde{\mathrm{q}}(w_{0:i})$ satisfies the conditions described in (117). This indicates that the hardmax attention is dispensable for our constructive proof.

# P Extension to Architecture with The QK Normalization

The QK normalization (Dehghani et al., 2023) applies the layer normalization (Ba et al., 2016) individually to both the query and key vectors to stabilize training. Specifically, concerning calculating attention scores, the QK normalization uses

$$\langle \mathrm{LN}(W_Q \mathbf{x}_{i_q}), \mathrm{LN}(W_K \mathbf{x}_{i_k}) \rangle \tag{299}$$

instead of

$$\langle W_Q \mathbf{x}_{i_q}, W_K \mathbf{x}_{i_k} \rangle, \tag{300}$$

where $\mathrm{LN}(\cdot)$ is the layer normalization (Ba et al., 2016). Specifically, the layer normalization parameterized by $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^{d_{\text{model}}}$ applies an affine transformation to $\mathbf{y} \in \mathbb{R}^{d_{\text{model}}}$ as follows:

$$\mathrm{LN}(\mathbf{y}) = \boldsymbol{\gamma} \odot \frac{\mathbf{y} - \mu(\mathbf{y})\mathbf{1}}{\sigma(\mathbf{y})} + \boldsymbol{\beta}, \tag{301}$$

where

$$\mu(\mathbf{y}) = \frac{1}{d_{\text{model}}} \sum_{d=1}^{d_{\text{model}}} y_d, \tag{302}$$

$$\sigma(\mathbf{y}) = \sqrt{\frac{1}{d_{\text{model}}} \sum_{d=1}^{d_{\text{model}}} (y_d - \mu(\mathbf{y}))^2}. \tag{303}$$

In this section, we show in two steps that the QK normalization can be incorporated into our constructive proof:

1. We give a proof that the layer normalization and the RMS layer normalization are equivalent when they are incorporated into the QK normalization with respect to their expressive power.

2. We show that our theoretical results also hold even when the QK normalization with the RMS layer normalization is incorporated into the architecture.

For clarity, denote the QK normalization with the layer normalization by `QK-LN` and the QK normalization with the RMS layer normalization by `QK-RMSLN`.

## P.1 Equivalence of the layer normalization and the RMS layer normalization under the QK normalization

We give a proof that for any attention layer with `QK-LN`, there exists an attention layer with `QK-RMSLN` that produces the same output (Lemma 11). Similarly, we also show that the converse holds: for any attention layer with `QK-RMSLN`, there exists an attention layer with `QK-LN` that produces the same output (Lemma 12). Note that it is sufficient to show the existence of a network that outputs the same attention scores.

**Lemma 11.** *For any attention layer with* `QK-LN`*, there exists an attention layer with* `QK-RMSLN` *that produces the same output for any given input.*

*Proof.* Assume the attention layer with `QK-LN` parameterized by $\boldsymbol{\beta}_Q, \boldsymbol{\gamma}_Q, \boldsymbol{\beta}_K, \boldsymbol{\gamma}_K$,

$$W_Q = \begin{bmatrix} \mathbf{w}_{Q,1}^\top \\ \vdots \\ \mathbf{w}_{Q,d_{\text{model}}}^\top \end{bmatrix}, W_K = \begin{bmatrix} \mathbf{w}_{K,1}^\top \\ \vdots \\ \mathbf{w}_{K,d_{\text{model}}}^\top \end{bmatrix}. \tag{304}$$

Then, the attention layer with QK-RMSLN parameterized by $\boldsymbol{\beta}'_Q = \boldsymbol{\beta}_Q, \boldsymbol{\gamma}'_Q = \boldsymbol{\gamma}_Q, \boldsymbol{\beta}'_K = \boldsymbol{\beta}_K, \boldsymbol{\gamma}'_K = \boldsymbol{\gamma}_K$,

$$W'_Q = \begin{bmatrix} \mathbf{w}_{Q,1}^\top - \left(\frac{1}{d_{\text{model}}} \sum_{d=1}^{d_{\text{model}}} \mathbf{w}_{Q,d}^\top\right) \\ \vdots \\ \mathbf{w}_{Q,d_{\text{model}}}^\top - \left(\frac{1}{d_{\text{model}}} \sum_{d=1}^{d_{\text{model}}} \mathbf{w}_{Q,d}^\top\right) \end{bmatrix}, \tag{305}$$

$$W'_K = \begin{bmatrix} \mathbf{w}_{K,1}^\top - \left(\frac{1}{d_{\text{model}}} \sum_{d=1}^{d_{\text{model}}} \mathbf{w}_{K,d}^\top\right) \\ \vdots \\ \mathbf{w}_{K,d_{\text{model}}}^\top - \left(\frac{1}{d_{\text{model}}} \sum_{d=1}^{d_{\text{model}}} \mathbf{w}_{K,d}^\top\right) \end{bmatrix} \tag{306}$$

produces the same attention scores. The reasons are detailed below: we obtain the query vector

$$\begin{aligned} & W'_Q \mathbf{x}_{i_q} \\ &= \begin{bmatrix} \mathbf{w}_{Q,1}^\top - \left(\frac{1}{d_{\text{model}}} \sum_{d=1}^{d_{\text{model}}} \mathbf{w}_{Q,d}^\top\right) \\ \vdots \\ \mathbf{w}_{Q,d_{\text{model}}}^\top - \left(\frac{1}{d_{\text{model}}} \sum_{d=1}^{d_{\text{model}}} \mathbf{w}_{Q,d}^\top\right) \end{bmatrix} \mathbf{x}_{i_q} \\ &= \begin{bmatrix} \mathbf{w}_{Q,1}^\top \mathbf{x}_{i_q} \\ \vdots \\ \mathbf{w}_{Q,d_{\text{model}}}^\top \mathbf{x}_{i_q} \end{bmatrix} - \begin{bmatrix} \frac{1}{d_{\text{model}}} \sum_{d=1}^{d_{\text{model}}} \mathbf{w}_{Q,d}^\top \mathbf{x}_{i_q} \\ \vdots \\ \frac{1}{d_{\text{model}}} \sum_{d=1}^{d_{\text{model}}} \mathbf{w}_{Q,d}^\top \mathbf{x}_{i_q} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{w}_{Q,1}^\top \mathbf{x}_{i_q} \\ \vdots \\ \mathbf{w}_{Q,d_{\text{model}}}^\top \mathbf{x}_{i_q} \end{bmatrix} - \mathbf{1}\mu\left(\begin{bmatrix} \mathbf{w}_{Q,1}^\top \mathbf{x}_{i_q} \\ \vdots \\ \mathbf{w}_{Q,d_{\text{model}}}^\top \mathbf{x}_{i_q} \end{bmatrix}\right). \end{aligned} \tag{307}$$

Here, since $\text{RMS}\left(\mathbf{y} - \mathbf{1}\mu(\mathbf{y})\right) = \sigma(\mathbf{y})$ holds,

$$\text{LN}_{\text{RMS}}(\mathbf{y} - \mathbf{1}\mu(\mathbf{y})) = \text{LN}(\mathbf{y}) \tag{308}$$

holds for any $\mathbf{y} \in \mathbb{R}^{d_{\text{model}}}$. Therefore,

$$\text{LN}_{\text{RMS}}(W'_Q \mathbf{x}_{i_q}) = \text{LN}(W_Q \mathbf{x}_{i_q}) \tag{309}$$

holds. Similarly,

$$\text{LN}_{\text{RMS}}(W'_K \mathbf{x}_{i_k}) = \text{LN}(W_K \mathbf{x}_{i_k}) \tag{310}$$

also holds, indicating that

$$\begin{aligned} & \langle \text{LN}_{\text{RMS}}(W'_Q \mathbf{x}_{i_q}), \text{LN}_{\text{RMS}}(W'_K \mathbf{x}_{i_k}) \rangle \\ &= \langle \text{LN}(W_Q \mathbf{x}_{i_q}), \text{LN}(W_K \mathbf{x}_{i_k}) \rangle. \end{aligned} \tag{311}$$

□

**Lemma 12.** *For any attention layer with* QK-RMSLN*, there exists an attention layer with* QK-LN *that produces the same output for any given input.*

*Proof.* Assume an attention layer with QK-RMSLN parameterized by $\boldsymbol{\beta}_Q, \boldsymbol{\gamma}_Q, \boldsymbol{\beta}_K, \boldsymbol{\gamma}_K$,

$$W_Q = \begin{bmatrix} \mathbf{w}_{Q,1}^\top \\ \vdots \\ \mathbf{w}_{Q,d_{\text{model}}}^\top \end{bmatrix}, \quad W_K = \begin{bmatrix} \mathbf{w}_{K,1}^\top \\ \vdots \\ \mathbf{w}_{K,d_{\text{model}}}^\top \end{bmatrix}. \tag{312}$$

Then, QK-LN parameterized by

$$\boldsymbol{\beta}''_Q = \begin{bmatrix} \boldsymbol{\beta}_Q \\ -\boldsymbol{\beta}_Q \\ \mathbf{0} \end{bmatrix}, \boldsymbol{\gamma}''_Q = \sqrt{\frac{2}{3}} \begin{bmatrix} \boldsymbol{\gamma}_Q \\ \boldsymbol{\gamma}_Q \\ \mathbf{1} \end{bmatrix}, \tag{313}$$

$$\boldsymbol{\beta}''_K = \begin{bmatrix} \boldsymbol{\beta}_K \\ \mathbf{0} \\ -\boldsymbol{\beta}_K \end{bmatrix}, \boldsymbol{\gamma}''_K = \sqrt{\frac{2}{3}} \begin{bmatrix} \boldsymbol{\gamma}_K \\ \mathbf{1} \\ \boldsymbol{\gamma}_K \end{bmatrix}, \tag{314}$$

$$W''_Q = \begin{bmatrix} \mathbf{w}_{Q,1}^\top \\ \vdots \\ \mathbf{w}_{Q,d_{\text{model}}}^\top \\ -\mathbf{w}_{Q,1}^\top \\ \vdots \\ -\mathbf{w}_{Q,d_{\text{model}}}^\top \\ \mathbf{0}^\top \\ \vdots \\ \mathbf{0}^\top \end{bmatrix}, W''_K = \begin{bmatrix} \mathbf{w}_{K,1}^\top \\ \vdots \\ \mathbf{w}_{K,d_{\text{model}}}^\top \\ \mathbf{0}^\top \\ \vdots \\ \mathbf{0}^\top \\ -\mathbf{w}_{K,1}^\top \\ \vdots \\ -\mathbf{w}_{K,d_{\text{model}}}^\top \end{bmatrix} \tag{315}$$

produces the same attention scores. The reasons are detailed below: we obtain the query vector

$$W''_Q \mathbf{x}_{i_q} = \begin{bmatrix} \mathbf{w}_{Q,1}^\top \mathbf{x}_{i_q} \\ \vdots \\ \mathbf{w}_{Q,d_{\text{model}}}^\top \mathbf{x}_{i_q} \\ -\mathbf{w}_{Q,1}^\top \mathbf{x}_{i_q} \\ \vdots \\ -\mathbf{w}_{Q,d_{\text{model}}}^\top \mathbf{x}_{i_q} \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{316}$$

Here, since $\mu(W''_Q \mathbf{x}_{i_q}) = 0$, the results of applying the layer normalization and the RMS layer normal-

ization to this vector are identical; that is,

$$
\begin{aligned}
&\mathrm{LN}(W''_Q \mathbf{x}_{i_q}) \\
&= \mathrm{LN}_{\mathrm{RMS}}(W''_Q \mathbf{x}_{i_q}) \\
&= \boldsymbol{\gamma}''_Q \odot \frac{1}{\sqrt{\frac{2\|W_Q \mathbf{x}_{i_q}\|_2^2}{3d_{\mathrm{model}}}}} \begin{bmatrix} \mathbf{w}_{Q,1}^\top \mathbf{x}_{i_q} \\ \vdots \\ \mathbf{w}_{Q,d_{\mathrm{model}}}^\top \mathbf{x}_{i_q} \\ -\mathbf{w}_{Q,1}^\top \mathbf{x}_{i_q} \\ \vdots \\ -\mathbf{w}_{Q,d_{\mathrm{model}}}^\top \mathbf{x}_{i_q} \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \boldsymbol{\beta}''_Q \\
&= \begin{bmatrix} \boldsymbol{\gamma}_Q \odot \frac{1}{\sqrt{\frac{\|W_Q \mathbf{x}_{i_q}\|_2^2}{d_{\mathrm{model}}}}} \begin{bmatrix} \mathbf{w}_{Q,1}^\top \mathbf{x}_{i_q} \\ \vdots \\ \mathbf{w}_{Q,d_{\mathrm{model}}}^\top \mathbf{x}_{i_q} \end{bmatrix} + \boldsymbol{\beta}_Q \\ -\boldsymbol{\gamma}_Q \odot \frac{1}{\sqrt{\frac{\|W_Q \mathbf{x}_{i_q}\|_2^2}{d_{\mathrm{model}}}}} \begin{bmatrix} \mathbf{w}_{Q,1}^\top \mathbf{x}_{i_q} \\ \vdots \\ \mathbf{w}_{Q,d_{\mathrm{model}}}^\top \mathbf{x}_{i_q} \end{bmatrix} - \boldsymbol{\beta}_Q \\ \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \end{bmatrix} \\
&= \begin{bmatrix} \mathrm{LN}_{\mathrm{RMS}}(W_Q \mathbf{x}_{i_q}) \\ -\mathrm{LN}_{\mathrm{RMS}}(W_Q \mathbf{x}_{i_q}) \\ \mathbf{0} \end{bmatrix}.
\end{aligned} \tag{317}
$$

Similarly,

$$
\mathrm{LN}(W''_K \mathbf{x}_{i_k}) = \begin{bmatrix} \mathrm{LN}_{\mathrm{RMS}}(W_K \mathbf{x}_{i_k}) \\ \mathbf{0} \\ -\mathrm{LN}_{\mathrm{RMS}}(W_K \mathbf{x}_{i_k}) \end{bmatrix}, \tag{318}
$$

indicating

$$
\begin{aligned}
&\langle \mathrm{LN}(W''_Q \mathbf{x}_{i_q}), \mathrm{LN}(W''_K \mathbf{x}_{i_k}) \rangle \\
&= \langle \mathrm{LN}_{\mathrm{RMS}}(W_Q \mathbf{x}_{i_q}), \mathrm{LN}_{\mathrm{RMS}}(W_K \mathbf{x}_{i_k}) \rangle.
\end{aligned} \tag{319}
$$

□

## P.2 Incorporating the QK normalization with the RMS layer normalization to our constructive proof

We use the attention layers for two purposes in our constructive proofs: (i) used to create pseudo positional encoding $\begin{bmatrix} \cos\phi(i) & \sin\phi(i) \end{bmatrix}^\top$ and depth vectors $\begin{bmatrix} \cos\theta(\mathrm{d}) & \sin\theta(\mathrm{d}) \end{bmatrix}^\top$ and (ii) used as an approximation of hardmax attention to focus on a single token. In the following sections, we show how to incorporate QK normalization into our constructive proofs.

**(i) When used to create positional and depth vectors**

When the attention layers are used to create positional vectors or depth vectors, an attention score of $a$ is assigned to <bos> and 0 to other tokens. We then show that this operation can be implemented also in the architecture with the QK normalization.

We omit the unnecessary dimensions of input vector $\mathbf{x}_i^{(\ell)}$ in this layer as follows:

$$
\mathbf{x}_i^{(\ell)} = \begin{bmatrix} \vdots \\ s_i \\ 1 \\ \vdots \end{bmatrix}. \tag{320}
$$

Then, the attention layer with QK-RMSLN parameterized by

$$
\boldsymbol{\beta}_Q^{(\ell)} = \mathbf{0}, \boldsymbol{\gamma}_Q^{(\ell)} = \sqrt{\frac{1}{d_{\mathrm{model}}}} \mathbf{1}, \tag{321}
$$

$$
\boldsymbol{\beta}_K^{(\ell)} = \mathbf{0}, \boldsymbol{\gamma}_K^{(\ell)} = a\sqrt{\frac{1}{d_{\mathrm{model}}}} \mathbf{1}, \tag{322}
$$

$$
W_Q^{(\ell)} = \begin{bmatrix} \cdots & 0 & 1 & \cdots \\ & \vdots & \vdots & \end{bmatrix}, \tag{323}
$$

$$
W_K^{(\ell)} = \begin{bmatrix} \cdots & 1 & 0 & \cdots \\ & \vdots & \vdots & \end{bmatrix} \tag{324}
$$

produces the desired attention scores. This is because

$$
\begin{aligned}
&\mathrm{LN}_{\mathrm{RMS}}\left(W_Q^{(\ell)} \mathbf{x}_{i_q}^{(\ell)}\right) \\
&= \mathrm{LN}_{\mathrm{RMS}}\left(\begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix}\right) \\
&= \sqrt{\frac{1}{d_{\mathrm{model}}}} \mathbf{1} \odot \begin{bmatrix} \sqrt{d_{\mathrm{model}}} \\ \mathbf{0} \end{bmatrix} + \mathbf{0} \\
&= \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix},
\end{aligned} \tag{325}
$$

$$
\begin{aligned}
&\mathrm{LN}_{\mathrm{RMS}}\left(W_K^{(\ell)} \mathbf{x}_{i_k}^{(\ell)}\right) \\
&= \mathrm{LN}_{\mathrm{RMS}}\left(\begin{bmatrix} s_{i_k} \\ \mathbf{0} \end{bmatrix}\right) \\
&= a\sqrt{\frac{1}{d_{\mathrm{model}}}} \mathbf{1} \odot \begin{bmatrix} \sqrt{d_{\mathrm{model}}} \cdot s_{i_k} \\ \mathbf{0} \end{bmatrix} + \mathbf{0} \\
&= \begin{bmatrix} s_{i_k} \cdot a \\ \mathbf{0} \end{bmatrix},
\end{aligned} \tag{326}
$$

indicating

$$\left\langle \mathrm{LN}_{\mathrm{RMS}}\left(W_K^{(\ell)}\mathbf{x}_{i_k}^{(\ell)}\right), \mathrm{LN}_{\mathrm{RMS}}\left(W_Q^{(\ell)}\mathbf{x}_{i_q}^{(\ell)}\right)\right\rangle = s_{i_k}\cdot a = \begin{cases} a & \text{if } w_{i_k} = \texttt{<bos>} \\ 0 & \text{otherwise} \end{cases}. \tag{327}$$

**(ii) When used as an approximation of hardmax attention**

Here, we show that even when the QK normalization with the RMS layer normalization is incorporated, any changes to the attention scores can be avoided: that is, we can fix the 2-norm of the query/key vectors to prevent changes in the attention scores. In our proofs, we only use the following values: $o_i, s_i, 1, \cos(\cdot), \sin(\cdot), \mathrm{q}_i$. Except for $\mathrm{q}_i$, by assigning the complementary values described below to the new dimensions of the query/key vector, we can fix the 2-norm. Here, we refer to $y$ and $\bar{y}$ as the complementary values if and only if $y^2 + \bar{y}^2$ is a constant. Here, $o_i$ and $s_i$ are complementary values, so are $\cos(\cdot)$ and $\sin(\cdot)$. This is because $o_i^2 + s_i^2 = 1$ and $\cos^2(\cdot) + \sin^2(\cdot) = 1$ hold. For example, if we set the attention parameters $W_Q, W_K$ to satisfy

$$W_Q\mathbf{x}_{i_q} = \begin{bmatrix} o_{i_q} \\ \cos\phi(i_q) \\ \mathbf{0} \end{bmatrix}, W_K\mathbf{x}_{i_k} = \begin{bmatrix} s_{i_k} \\ 1 \\ \mathbf{0} \end{bmatrix}, \tag{328}$$

by modifying them into

$$W'_Q\mathbf{x}_{i_q} = \begin{bmatrix} o_{i_q} \\ \cos\phi(i_q) \\ s_{i_q} \\ \sin\phi(i_q) \\ 0 \\ \mathbf{0} \end{bmatrix}, W'_K\mathbf{x}_{i_k} = \begin{bmatrix} s_{i_k} \\ 1 \\ 0 \\ 0 \\ o_{i_k} \\ \mathbf{0} \end{bmatrix}, \tag{329}$$

we can fix the 2-norm of the query/key vectors. This is because

$$\|W'_Q\mathbf{x}_{i_q}\|_2^2 = o_{i_q}^2 + s_{i_q}^2 + \cos^2\phi(i_q) + \sin^2\phi(i_q) = 2, \tag{330}$$

$$\|W'_K\mathbf{x}_{i_k}\|_2^2 = s_{i_k}^2 + o_{i_k}^2 + 1^2 = 2. \tag{331}$$

Therefore, by setting $\boldsymbol{\beta}_Q = \boldsymbol{\beta}_K = \mathbf{0}, \boldsymbol{\gamma}_Q = \frac{\|W'_Q\mathbf{x}_{i_q}\|_2}{\sqrt{d_{\mathrm{model}}}}\mathbf{1}$ and $\boldsymbol{\gamma}_K = \frac{\|W'_K\mathbf{x}_{i_k}\|_2}{\sqrt{d_{\mathrm{model}}}}\mathbf{1}$, we obtain

$$\mathrm{LN}_{\mathrm{RMS}}\left(W'_Q\mathbf{x}_{i_q}\right) = \boldsymbol{\gamma}_Q \odot \frac{\sqrt{d_{\mathrm{model}}}}{\|W'_Q\mathbf{x}_{i_q}\|_2}\left(W'_Q\mathbf{x}_{i_q}\right) = W'_Q\mathbf{x}_{i_q}, \tag{332}$$

$$\mathrm{LN}_{\mathrm{RMS}}\left(W'_K\mathbf{x}_{i_k}\right) = \boldsymbol{\gamma}_K \odot \frac{\sqrt{d_{\mathrm{model}}}}{\|W'_K\mathbf{x}_{i_k}\|_2}\left(W'_K\mathbf{x}_{i_k}\right) = W'_K\mathbf{x}_{i_k}, \tag{333}$$

indicating that the attention scores generated by the attention layer with the QK normalization are identical to those generated without the QK normalization.

In contrast, $\mathrm{q}_i$ is used in the key vector defined in Appendix G.5, and it is hard to fix the 2-norm. However, by setting $\boldsymbol{\beta}_K = 0, \boldsymbol{\gamma}_K = \frac{1}{\sqrt{d_{\mathrm{model}}}}\mathbf{1}$, we obtain

$$\mathrm{LN}_{\mathrm{RMS}}\left(W_K^{(5)}\mathbf{x}_{i_k}^{(5)}\right) = \mathrm{LN}_{\mathrm{RMS}}\left(\begin{bmatrix} -\mathrm{q}_{i_k} \\ \mathrm{q}_0 \cdot s_{i_k} \\ \mathbf{0} \end{bmatrix}\right) = \begin{cases} \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ \mathbf{0} \end{bmatrix} & \text{if } i_k = 0 \\ \begin{bmatrix} -1 \\ 0 \\ \mathbf{0} \end{bmatrix} & \text{if } \mathrm{q}_{i_k} > 0 \\ \begin{bmatrix} 1 \\ 0 \\ \mathbf{0} \end{bmatrix} & \text{if } \mathrm{q}_{i_k} < 0 \end{cases}, \tag{334}$$

which leads to the same result.

## Q Rationale behind Architectural Modification

Although the architecture adopted in Yao et al. (2021) uses the conventional layer normalization, we adopt an architecture with the RMS layer normalization. This is not only because recent models such as LLama (Touvron et al., 2023a) and Llama 2 (Touvron et al., 2023b) adopt the RMS layer normalization but also because we try to make our constructive proofs more concise. In this section,

we show that this change does not affect the critical aspects of our proofs; in other words, we give a proof that any transformation achievable with the RMS layer normalization can be achieved with the conventional layer normalization.

**Lemma 13.** *For any feed-forward network with the RMS layer normalization and a hidden size of $d_{\text{model}}$, there exists a feed-forward network with the layer normalization and a hidden size of $2d_{\text{model}}$ such that their outputs are identical.*

*Proof.* Consider the feed-forward network layer with the RMS layer normalization parameterized by $W_1, W_2, \boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, the output becomes

$$\begin{aligned} &W_2[\mathrm{LN}_{\mathrm{RMS}}(W_1\mathbf{x})]_+ \\ &= \frac{1}{\mathrm{RMS}\left(W_1\mathbf{x}\right)} W_2[\boldsymbol{\gamma} \odot (W_1\mathbf{x}) + \boldsymbol{\beta}]_+. \end{aligned} \tag{335}$$

This output is realized by the feed-forward network layer with the layer normalization parameterized by $W_1' = \begin{bmatrix} W_1 \\ -W_1 \end{bmatrix}, W_2' = \begin{bmatrix} W_2 & O \end{bmatrix}, \boldsymbol{\beta}' = \begin{bmatrix} \boldsymbol{\beta} \\ \mathbf{0} \end{bmatrix}$ and $\boldsymbol{\gamma}' = \begin{bmatrix} \boldsymbol{\gamma} \\ \mathbf{1} \end{bmatrix}$. This is because

$$\begin{aligned} &W_2' \left[\mathrm{LN}\left(W_1'\mathbf{x}\right)\right]_+ \\ &= \frac{1}{\sigma\left(W_1'\mathbf{x}\right)} \begin{bmatrix} W_2 & O \end{bmatrix} \begin{bmatrix} [\boldsymbol{\gamma} \odot (W_1\mathbf{x}) + \boldsymbol{\beta}]_+ \\ [(-W_1\mathbf{x})]_+ \end{bmatrix}. \\ &= \frac{1}{\mathrm{RMS}\left(W_1'\mathbf{x}\right)} W_2[\boldsymbol{\gamma} \odot (W_1\mathbf{x}) + \boldsymbol{\beta}]_+. \\ &\quad \text{(because } \mu\left(W_1'\mathbf{x}\right) = 0) \end{aligned} \tag{336}$$

Here,

$$\begin{aligned} &\mathrm{RMS}\left(W_1'\mathbf{x}\right) \\ &= \sqrt{\frac{1}{2d_{\text{model}}}(\|W_1\mathbf{x}\|_2^2 + \|-W_1\mathbf{x}\|_2^2)} \\ &= \sqrt{\frac{1}{d_{\text{model}}}}\|W_1\mathbf{x}\|_2 \\ &= \mathrm{RMS}\left(W_1\mathbf{x}\right), \end{aligned} \tag{337}$$

indicating that the two transformations produce the same outputs. □

# R Details of Experiments

## R.1 Full evaluation on $\texttt{Dyck}_k$

### Setup

The $\texttt{Dyck}_k$ and $\texttt{Shuffle-Dyck}_k$ language datasets are generated by $p_{\texttt{Dyck}_k}(\cdot; q, r, \boldsymbol{\pi})$ parameterized with $q = 0.5, r = 0.9, \boldsymbol{\pi} = \frac{1}{k}\mathbf{1}$ and $p_{\texttt{Shuffle-Dyck}_k}(\cdot; q, r, \boldsymbol{\pi}, \overline{\boldsymbol{\pi}})$ parameterized with $q = 0.3, r = 0.97, \boldsymbol{\pi} = \frac{1}{k}\mathbf{1}, \overline{\boldsymbol{\pi}} = \frac{1}{k}\mathbf{1}$, respectively. Compared to $\texttt{Dyck}_k$, we set the smaller value for $q$ and the larger value for $r$ in the case of $\texttt{Shuffle-Dyck}_k$ for two reasons: (i) to avoid the situation where all types remain unclosed in the later positions, making the task trivial and (ii) to prevent the generation of an excessive number of short sequences due to the small $q$.

Following Yao et al. (2021), we set $n_{\max} = 700$ and $d_{\text{model}} = 30$, and we truncated the sequences longer than $n_{\max}$. We generated $100,000$ sequences as training data, with an additional $10,000$ sequences (equivalent to $10\%$ of the training data) used for both validation and test datasets. Note that for the test data, we create out-of-distribution (OOD) sequences with respect to length, generating sequences up to a maximum length of $1.2 \times n_{\max}$.

We conducted experiments by varying the presence of <bos> ($\{\texttt{BOS}, \texttt{NoBOS}\}$), the presence of positional encoding ($\{\texttt{PE}, \texttt{NoPE}\}$), the number of brackets types ($\{1, 2, 4, 8, 16\}$ for $\texttt{Dyck}_k$ and $\{2, 4, 8, 16\}$ for $\texttt{Shuffle-Dyck}_k$), and the number of layers ($\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$). Here, each 10-layer model has a size of $0.05$M parameters.

We set the learning rate candidates to $\{3\text{e}{-3}, 3\text{e}{-4}\}$ and evaluated the performance of the model that achieved the lowest validation loss. We report the average performance over $5$ runs with different random seeds.

### Metric

Following Hewitt et al. (2020), Yao et al. (2021), we evaluated the model performance using the conditional probability of outputting the correct closing brackets on test data. In addition, we also reported the TV distance from the true language generation process.

The test data contains sequences whose length is up to $1.2 \times n_{\max}$. We regard tokens at position $i \leq n_{\max}$ as in-distribution (ID) data and tokens at position $n_{\max} < i \leq 1.2 \times n_{\max}$ as out-of-distribution (OOD) data, thereby we evaluate the generalization ability with respect to sequence length.

Figure 8 and 9 show the average test accuracy of generating the correct closed bracket on $\texttt{Dyck}_k$ and $\texttt{Shuffle-Dyck}_k$, respectively. Moreover, Figure 10 and 11 show the average test TV distance on $\texttt{Dyck}_k$ and $\texttt{Shuffle-Dyck}_k$, respectively.

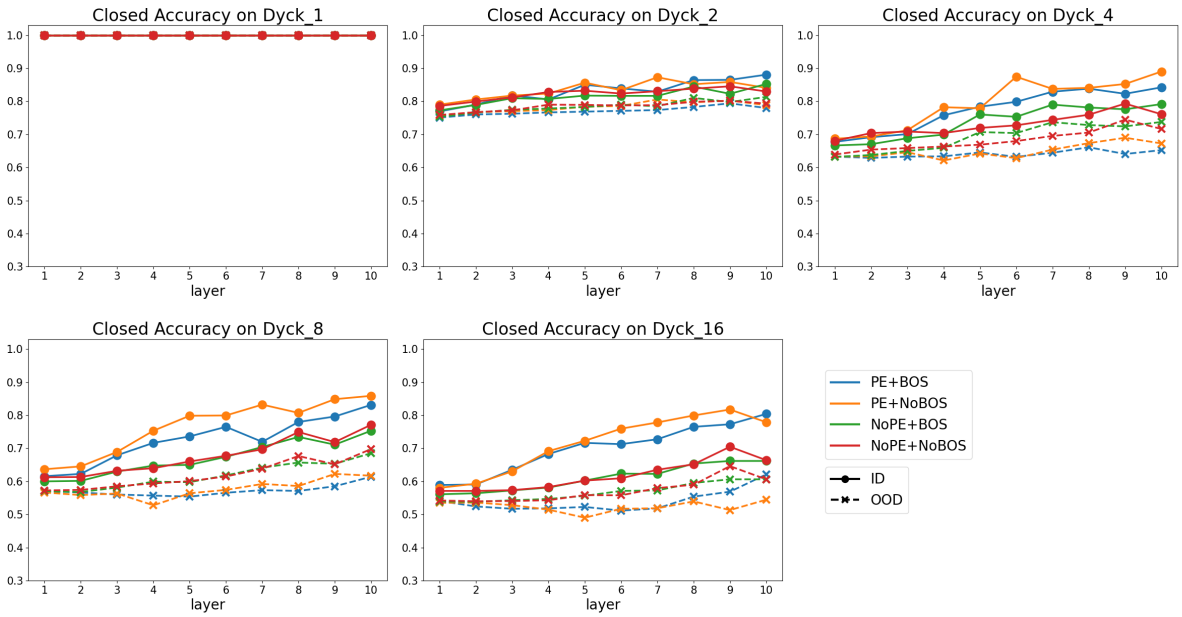


Figure 8: Test accuracy over 5 runs of generating the correct closed brackets on $\texttt{Dyck}_k (k \in \{1, 2, 4, 8, 16\})$. The solid lines represent the results for in-distribution data ($n \leq 700$), while the dashed lines represent the results for out-of-distribution data ($700 < n \leq 840$).

| **Hyperparameter** | **Value** |
|---|---|
| Model parameters | |
| Number of attention heads | 1 |
| Embedding dimension $d_{\text{model}}$ | 30 |
| Use of bias terms | `False` |
| Affine Transformation in the RMS layer normalization | `True` |
| Window size | $1,024$ |
| Activation function | ReLU |
| Training parameters | |
| Dropout rate | 0.0 |
| Batch size | 16 |
| Learning rate | $\{3e{-}3, 3e{-}4\}$ |
| Gradient accumulation steps | 2 |
| Weight decay | 0.0 |
| Adam parameters $(\beta_1, \beta_2)$ | $(0.9, 0.999)$ |
| Maximum iterations | $3,000$ |
| Warmup iterations | 0 |
| Learning rate decay | `False` |

Table 4: Hyperparameter configuration for experiments on the $\texttt{Dyck}_k$ language.

### R.2 Evaluation on natural language datasets

In Section 5.2, to investigate how our modification on layer normalization position affects the model performance, we compared perplexity on natural language datasets across four positions: `Post-LN`, `Pre-LN`, `No-LN`, and `FFN-LN` because Wang et al. (2019) and Xiong et al. (2020) empirically showed that layer normalization position significantly affects the model performance. We used two natural language datasets, WikiText-103 [2] (Merity et al., 2016), a common English dataset that contains over 100 million tokens extracted from the articles on Wikipedia, and OpenWebText [3] (Gokaslan et al., 2019), a 30GB of common English dataset that contains HTML pages whose URLs are shared on Reddit. Here, we provide detailed experimental settings and other experimental results.

Generally, there are two types regarding the position of the layer normalization used in Transformer architectures. One is `Post-LN`, which is used in

[2]The WikiText-103 dataset is licensed under CC BY-SA 3.0, and we can freely use the content as long as we provide appropriate attribution. Our use of this dataset is consistent with the intended use. To the best of our knowledge, there is no specific step that checks whether personal information or offensive content is contained.

[3]The OpenWebText is licensed under Creative Commons CC0 license, and we can freely use the content. Our use of this dataset is consistent with the intended use. To the best of our knowledge, there is no specific step that checks whether personal information or offensive content is contained.

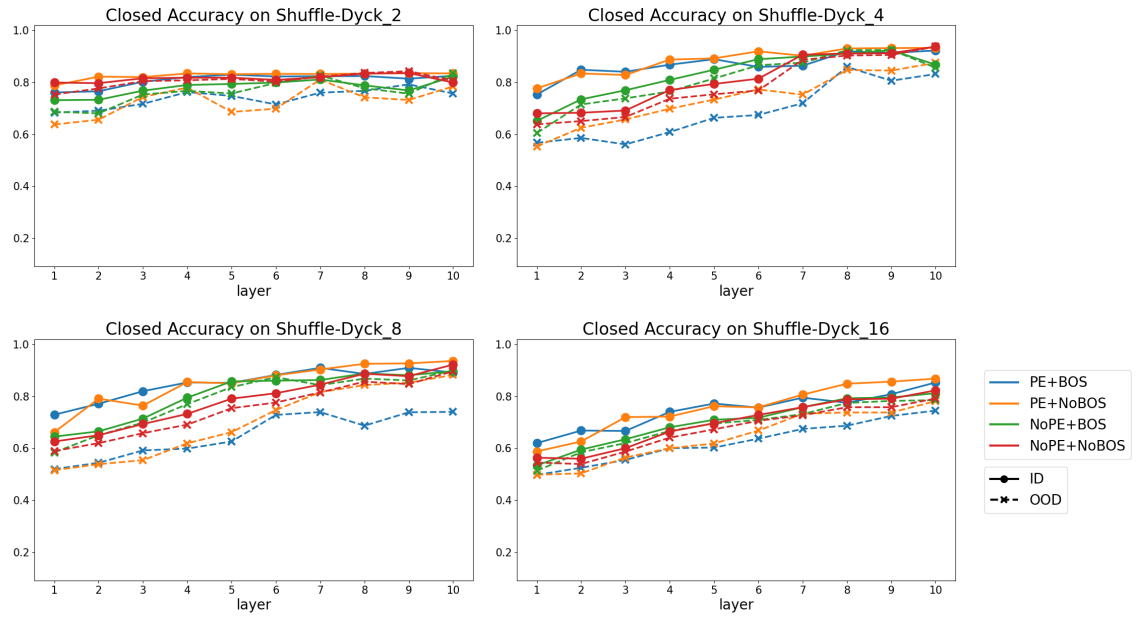


Figure 9: Test accuracy over 5 runs of generating the correct closed brackets on $\texttt{Shuffle-Dyck}_k (k \in \{2, 4, 8, 16\})$. The solid lines represent the results for in-distribution data ($n \leq 700$), while the dashed lines represent the results for out-of-distribution data ($700 < n \leq 840$).

models such as the original Transformer (Vaswani et al., 2017) and GPT (Radford et al., 2018), and the other is Pre-LN, which is used in models such as GPT-2 (Radford et al., 2019), GPT-3 (Brown et al., 2020), Llama (Touvron et al., 2023a), and Llama 2 (Touvron et al., 2023b).

We used the default split for WikiText-103: $103,227,021$ tokens from $28,475$ articles for training, $217,646$ tokens from 60 articles for validation, and $245,569$ tokens from 60 articles for test. In contrast, for OpenWebText, we used $0.5\%$ of the total data for the validation set following the approach of Fu et al. (2023), and similarly used $0.5\%$ for the test set.

We implemented the architecture based on nanoGPT[4], which is a small version of GPT and incorporates the GPT-2 tokenizer in the tiktoken library [5]. We add modifications to the position of the layer normalization. Regarding the hyperparameters, we used the default values except the values concerning the number of iterations: we modified the number of iterations to $20,000$, and accordingly, we also modified the number of iterations for learning-rate decay to $20,000$. Note that we adopt the QK normalization (Dehghani et al., 2023) to stabilize training. We use NVIDIA A100, and each experiment on WikiText-103 required approximately 40 GPU hours, while each experiment on OpenWebText required approximately 100 GPU hours. The values of the other hyperparameters are summarized in Table 5, and the decrease in training and validation loss is shown in Figure 12.

In addition, we empirically investigated the ability of length generalization with WikiText-103. We trained four models with FFN-LN (PE+BOS, PE+NoBOS, NoPE+BOS, and NoPE+NoBOS) on sequences of length 700 and tested the four models on sequences of length $1.2 \times n_{\max} (= 840)$. we modified the number of iterations from $20,000$ to $10,000$ because $10,000$ iterations are enough for models to converge.

We use NVIDIA A100, and each experiment on WikiText-103 required approximately 20 GPU hours. The values of the hyperparameters, except the sequence length and the number of iterations, are the same as summarized in Table 5. Figure 13 shows that the decrease in training and validation loss ($n_{\max} = 700$), and Table 6 shows the test perplexities on sequences of length $1.2 \times n_{\max} (= 840)$.

The decrease in training loss shows little difference between cases with and without explicit positional encoding. In contrast, the models without explicit positional encoding achieve much better test perplexity than those with explicit positional encoding. These results suggest that explicit posi-

[4] nanoGPT(https://github.com/karpathy/nanoGPT) is licensed under MIT License, and we can freely use, copy, modify, publish, and distribute.

[5] https://github.com/openai/tiktoken

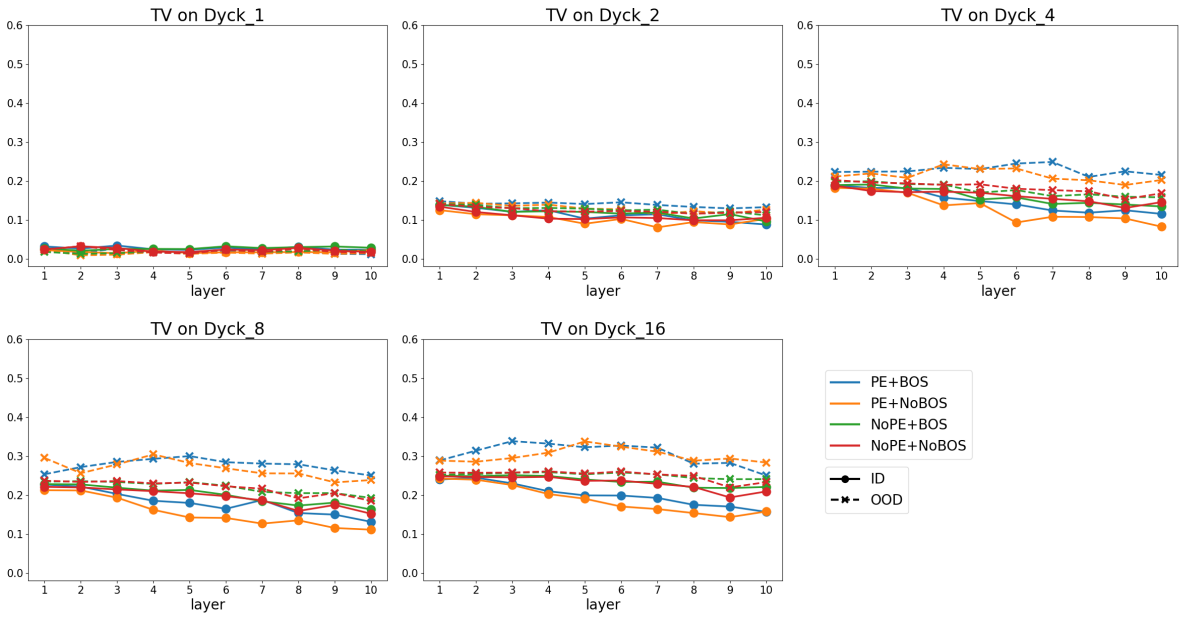


Figure 10: Average TV distance over 5 runs on $\texttt{Dyck}_k (k \in \{1, 2, 4, 8, 16\})$. The solid lines represent the results for in-distribution data ($n \leq 700$), while the dashed lines represent the results for out-of-distribution data ($700 < n \leq 840$).

| Hyperparameter | Value |
|---|---|
| Model parameters | |
| Number of layers $L$ | 12 |
| Number of attention heads | 12 |
| Embedding dimension $d_{\text{model}}$ | 768 |
| Use of bias terms | `False` |
| Window size | 1,024 |
| Activation function | gelu |
| Training parameters | |
| Dropout rate | 0.0 |
| Batch size | 12 |
| Gradient accumulation steps | 40 |
| Learning rate | 6e−4 |
| Minimum learning rate | 6e−5 |
| Weight decay | 1e−1 |
| Adam parameters $(\beta_1, \beta_2)$ | $(0.9, 0.95)$ |
| Maximum iterations | 20,000 |
| Warmup iterations | 2,000 |
| Learning rate decay iterations | 20,000 |

Table 5: Hyperparameter configuration for experiments on natural language datasets.

| Architecture | Test Perplexity |
|---|---|
| PE+BOS | 21.97 |
| PE+NoBOS | 22.41 |
| NoPE+BOS | 19.55 |
| NoPE+NoBOS | 19.43 |

Table 6: Perplexity on test dataset ($n \leq 1.2 \cdot n_{\max} = 840$) for the four models (PE+BOS, PE+NoBOS, NoPE+BOS, and NoPE+NoBOS) trained with sequences of $n_{\max} = 700$.

tional encoding might have a negative impact on the model's ability to generalize to longer sequences.

## S Further Discussion on Layer Normalization Position

A common explanation for the reason why the layer normalization leads to good performance is that the layer normalization stabilizes the output distribution. Recently, some studies have investigated how the position of the layer normalization affects the model performance.

Most of the recent models such as Llama (Touvron et al., 2023a), Llama 2 (Touvron et al., 2023b), GPT-2 (Radford et al., 2019), and GPT-3 (Brown et al., 2020) adopt `Pre-LN`, while the original Transformer architecurue (Vaswani et al., 2017) and GPT (Radford et al., 2018) adopt `Post-LN`. There are some studies supporting that `Pre-LN` outperforms `Post-LN`. However, there are also results indicat-

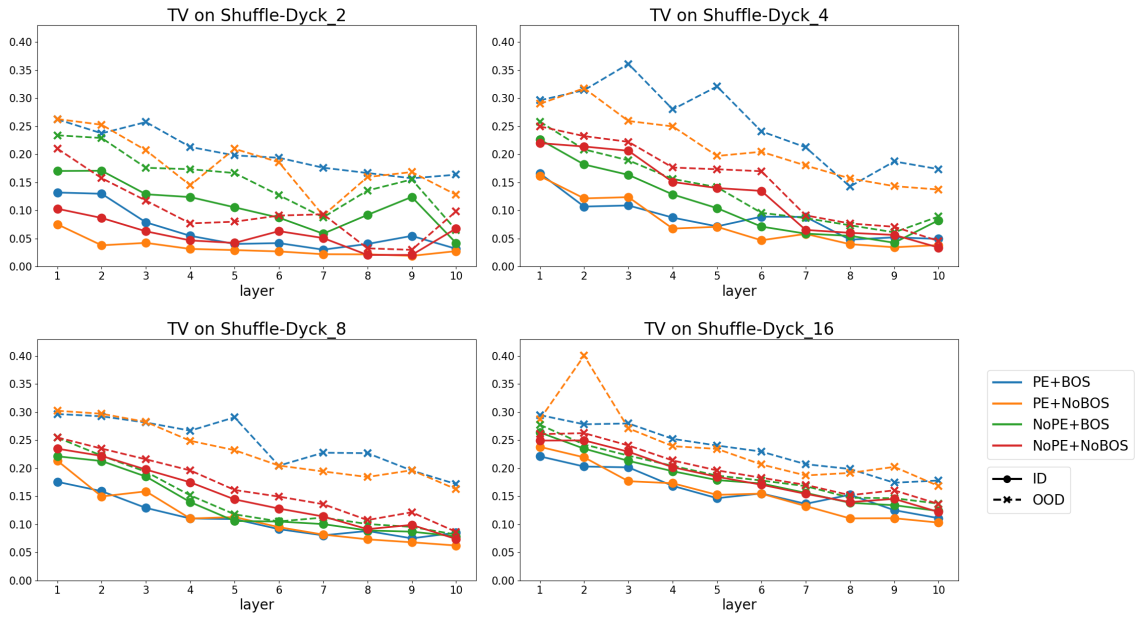


Figure 11: Average TV distance over 5 runs on $\texttt{Shuffle-Dyck}_k (k \in \{2, 4, 8, 16\})$. The solid lines represent the results for in-distribution data ($n \leq 700$), while the dashed lines represent the results for out-of-distribution data ($700 < n \leq 840$).

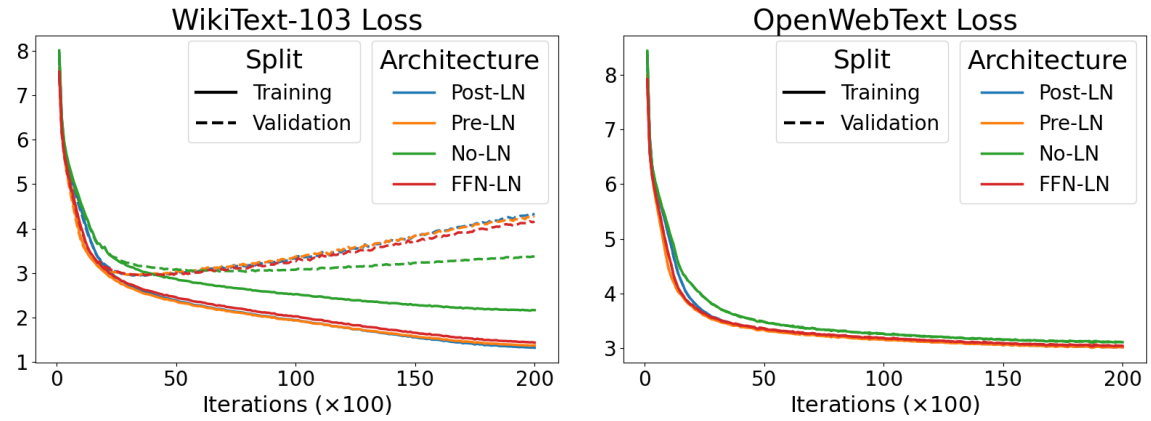


Figure 12: Results on natural language datasets. The transitions of training and validation losses are reported.

ing that Post-LN can outperform Pre-LN under specific conditions.

Xiong et al. (2020) analyzed the layer normalization from the perspective of mean-field theory and showed that Pre-LN provides more stable gradient after initialization compared to Post-LN. Xiong et al. (2020) also empirically showed that Pre-LN, unlike Post-LN, does not require a warmup phase and significantly reduces training time. In addition, Wang et al. (2019) suggested that Post-LN can have a higher risk of gradient vanishing and that in settings with a large number of layers, which are commonly seen in recent years, Pre-LN outperforms Post-LN.

In contrast, with respect to neural machine translation (NMT) task, Nguyen and Salazar (2019) showed that although Pre-LN contributes to training stability and better performance in low-resource settings, Post-LN shows superior performance in high-resource settings. Moreover, Mao et al. (2023) demonstrated that for zero-shot machine translation, Post-LN consistently outperforms Pre-LN. Furthermore, Shleifer et al. (2021) demonstrated that incorporating the layer normalization right before the second linear layer of the feed-forward network layer can effectively mitigate gradient explosion and vanishing, which are observed commonly in both Pre-LN and Post-LN setups.

Based on these results, we concluded that the optimal position of the layer normalization has not

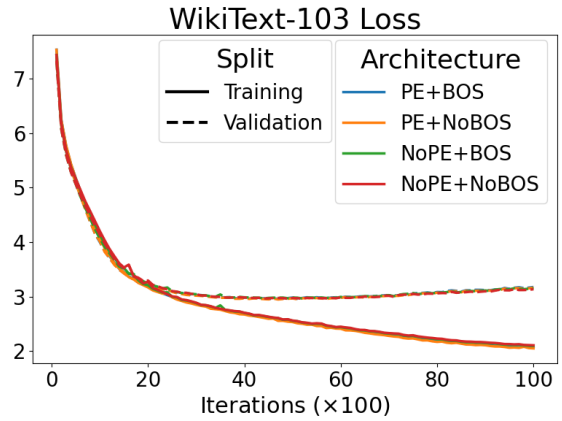


Figure 13: Results on WikiText-103 dataset. The transitions of training and validation losses are reported. Note that the four models (PE+BOS, PE+NoBOS, NoPE+BOS, and NoPE+NoBOS) are trained with sequences of length 700.

been established yet. Although the optimal position of the layer normalization remains unclear, in our experiments using the WikiText-103 and OpenWebText, we observed that the performance of Pre-LN, Post-LN, and FFN-LN consistently outperformed No-LN. Therefore, we concluded that the architecture used in our proof FFN-LN is competitive compared to other layer normalization positions, Pre-LN and Post-LN.