ON THE REPRESENTATION GAP BETWEEN MODERN RNNS AND TRANSFORMERS: THE CURSE OF MEMORY EFFICIENCY AND THE FIX OF IN-CONTEXT RETRIEVAL

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper investigates the limitations of Recurrent Neural Networks (RNNs) in algorithmic tasks, particularly in comparison with Transformers. Focusing on a reasoning task IsTree deciding whether a graph is a tree, we demonstrate that RNNs with $o(n)$ parameters, even with Chain-of-Thought (CoT), cannot solve this task for graphs with size n, unlike Transformers which can solve the task with CoT and only $O(\log n)$ bit parameters. Our experiments confirm this representation gap. To overcome this limitation, we propose augmenting RNNs with in-context retrieval capabilities, specifically using regular expressions. This enhancement enables RNNs to solve IsTree and other algorithmic problems in P, maintaining their memory efficiency and closing the gap with Transformers.

1 INTRODUCTION

Transformer models (Vaswani et al., 2017) have become the dominant choice of the backbone for large language models (LLMs). The core component of Transformers is its self-attention module, which allows the model to route information densely across the entire sequence. However, this design leads to high inference costs for modeling long sequences, including a memory cost that is at least linear in the sequence length due to the need for maintaining intermediate attention keys and values for each token, and a time cost quadratic in the sequence length for computing the attention score for each pair of tokens.

Recently, Recurrent Neural Networks (RNNs) have been an increasingly popular choice in sequence modeling tasks due to their ability to maintain a memory size constant in sequence length during inference, thus achieving a better scaling than Transformers. Successful examples include RWKV (Peng et al., 2023), RetNet (Sun et al., 2023), and Mamba (Gu & Dao, 2023). Most notably, Mamba can achieve competitive performance with Transformers on several sequence modeling tasks with linear time and constant memory complexity in sequence length.

The rise of these modern RNNs has led to an interest in understanding their capabilities and limitations compared to Transformers. Arora et al. (2023) conducted empirical studies to show that RNNs underperform Transformers on associative recall tasks that can naturally occur in language modeling. The most relevant work in this direction is Feng et al. (2023a), where they studied the representation power of language models with Chain-of-Thought (CoT) (Wei et al., 2023), a prompting technique that asks the model to generate many intermediate tokens before giving the final answer. They demonstrated a few examples of algorithmic tasks where Transformers can generate a specific sequence of CoT to solve the tasks, but RNNs cannot follow this sequence exactly. However, these studies do not exclude the possibility that, with a more careful design of architecture or a better curation of training data for CoT, RNNs may be able to achieve the same or even better performance than transformers. There is also a parallel thread of works Li et al. (2021; 2022) that tries to understand the optimization perspectives of RNNs, which is orthogonal to our work.

Our Contributions. In this paper, we consider a fairly general class of RNNs and take a step forward by revealing a fundamental limitation of RNNs in solving algorithmic tasks. More specifically, we showcase a minimal example of a reasoning task that RNNs and Transformers can have a significant representation gap:

$$\text{IsTree: Given an undirected graph } G \text{ of } n \text{ nodes, determine whether } G \text{ is a tree.}$$

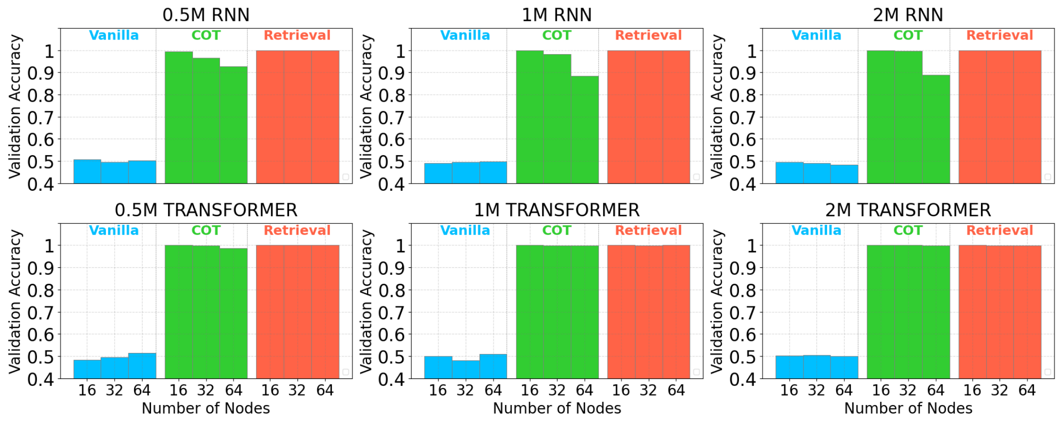

Figure 1: We train RNNs (Mamba) and Transformers (LLaMA 2 Touvron et al. (2023)) with a frozen word embedding and decoding head of three different model sizes (0.5M, 1M, 2M) on IsTree with three different sizes of graph (16, 32, 64) under three different setups. The graph is generated in the same way as in the proof of Lemma B.10. We train every model with at least 1M samples using Adam with a learning rate 1e-3 for the Transformer and 3e-4 for RNNs until convergence, extending to 5M samples if the model doesn't converge. **Vanilla** means the model directly predicts the label. **COT** means the model will generate a chain-of-thought process based on DFS (see Algorithm 1) before prediction. During inference, we will let the model generate the chain of thought until either generating YES or NO or until a max token limit and evaluate based on the final prediction regardless of the reasoning process. **Retrieval** means the model will generate the chain of search queries and reasoning before prediction (see Algorithm 2). We observe that (1) Both Transformer and RNNs can't solve the IsTree question without a chain of thought; (2) RNNs' performance with chain-of-thought decays quickly when the number of nodes increase, which is consistent with our theory; (3) All models reach almost perfect accuracy when enhanced with in-context retrieval.

We assume a standard and realistic assumption that the floating-point precision is $O(\log n)$ (Merrill & Sabharwal, 2023; Liu et al., 2023; Feng et al., 2023b), and prove that any constant-size RNNs are not able to solve IsTree with arbitrarily many nodes. We also show that even if we allow RNNs to do Chain-of-Thought (CoT) of any form, RNNs are still not able to solve IsTree. On the flip side, we show that there exists a constant-size Transformer that can perfectly solve IsTree with CoT, which is in line with previous works (Anonymous, 2023; Feng et al., 2023a; Liu et al., 2023) showing the algorithmic reasoning capabilities of Transformer. Our experiments also confirm that RNNs fail to solve IsTree for large n while Transformers can solve it with ease.

In essence, the above limitation of RNNs holds for all RNNs with $o(n)$-bit memory, and thus cannot be overcome without sacrificing their memory efficiency. We further show that allowing RNNs to invoke function calls to retrieve information in context can be a promising direction to address this limitation while keeping the RNN size constant. More specifically, we consider one of the simplest forms of retrieval: RNNs are allowed to search with a regular expression on the sequence that has been processed so far. We prove that this simple form of retrieval is sufficient to close the representation gap: RNNs augmented with this in-context retrieval capability can solve IsTree with arbitrarily many nodes. Beyond IsTree, we also prove that this is sufficient for RNNs to solve any polynomial time algorithmic problems.

2 The Representation Gap Between RNNs and Transformers

2.1 Definitions

Here we introduce definitions that are necessary for understanding our results. We refer to reader to Appendix A for full definitions.

Language Modeling. A vocabulary V is a finite set of tokens. A word embedding for a token is a specific vector in $\mathbb{R}^d$ that represents the token, and a position embedding for a token at a position in a sequence is a specific vector in $\mathbb{R}^d$ that represents the position. Given a sequence of tokens $\mathcal{S}$, an embedding function $\text{Emb}(\mathcal{S})$ maps each token to a vector in $\mathbb{R}^d$ by mixing word and position embeddings, resulting in a sequence of vectors. See Appendix A.3 for the specific choice of the embedding function in this paper to ease the comparison between RNNs and Transformers. A language model is defined as function $M : \cup_{l\geq 1}|V|^l \to \mathbb{P}_{|V|}$ that maps a prefix to the probability distribution over the next token, where $|V|$ is the vocabulary size, l is the sequence length, and $\mathbb{P}_{|V|}$ is the probability simplex over $|V|$. In deep learning, a language model is usually realized by first embedding the input tokens into vectors and then applying a neural network (such as Transformers and RNNs) to process the vectors and output the probability distribution.

Transformer. In the context of language modeling, given a sequence of tokens $\mathcal{S}$, a Transformer $T(\mathcal{S})$ is a neural network that produces the following outputs:

$$T(\mathcal{S}) = \sigma\left(W^{(E)}\left(f_L\left(\ldots f_1\left(\text{Emb}\left(\mathcal{S}\right)\right)\right)\right)\right)_{:,l}. \tag{1}$$

where σ is the column-wise softmax function, $W^{(E)}$ is a frozen predefined word embedding matrix, f_i is the i-th Transformer block. A Transformer block consists of a feed-forward layer and an attention layer, along with residual connections. See Definition A.4 for details. We call the i-th transformer block the i-th layer of the transformer.

Recurrent Neural Networks Recently there has been a lot of interest in the linear-time transformer, which replaces the full-attention calculation with linear-time alternatives. These variants are mostly special forms of recurrent neural networks that are parallelizable. We will define a general form of recurrent neural networks containing all of these variants; hence, our analysis will apply to all of them. A Recurrent Neural Network (RNN) is characterized by two functions: state transition function $\mathbf{t} : \mathbb{R}_p^\Lambda \times \mathbb{R}_p^d \to \mathbb{R}_p^\Lambda$ and output function $\mathbf{o} : \mathbb{R}_p^\Lambda \to \mathbb{R}_p^d$, where d is the dimension of the state. Let $\mathcal{S} \in |V|^l$ be the input sequence, the output of a recurrent neural network is defined as:

$$R(\mathcal{S}) = \sigma\left(W^{(E)}\mathbf{o}(s_n)\right)_{:,l}, \tag{2}$$

$$\forall k \in [n], s_k = \mathbf{t}(s_{k-1}, \text{Emb}(\mathcal{S})_{:,k}), \tag{3}$$

where $s_0 \in \mathbb{R}_p^\Lambda$ is a constant vector and $W^{(E)}$ is a frozen predefined word embedding matrix, which is the same one in the definition of Transformer. We can characterize the complexity of an RNN with the following three measures,

1. Parameter size P: the number of bit of parameters in $\mathbf{t}$ and $\mathbf{o}$.
2. State memory size M: the number of bits in memory required to simulate a fixed parameter RNN, which in this case is $\Lambda \times p$.
3. Circuit size C: the number of bit-wise arithmetic operations needed to calculate $\mathbf{t}$ and $\mathbf{o}$.

We further constrain ourselves to RNNs with the following properties, which still contain all the modern variants of linear-time transformers to the best of our knowledge.

Definition 2.1. We say that an RNN is regular if $\mathrm{P} = \Omega(\mathrm{M})$ and $\mathrm{C} = \tilde{\Theta}(\mathrm{P})$.

Chain of Thought. Given a language model M with vocabulary V and a sequence of tokens $\mathcal{S}_{\text{in}} \in |V|^{l_0}$, we define Chain of Thought (CoT) as the following process of sequence generation:

$$\mathcal{S}_0 = \mathcal{S}_{\text{in}}, \qquad s_i^{\text{next}} = \arg\max_{j \in V} M(\mathcal{S}_i)[j], \qquad \mathcal{S}_{i+1} = \mathcal{S}_i \oplus s_i^{\text{next}}, \forall i \geq 0, \tag{4}$$

The process terminates at $\mathcal{S}_i$ when $\arg\max_{j \in V} M(\mathcal{S}_i)[j]$ is YES or NO. We say that the language model can solve a task that expects answer YES or NO within T steps of CoT if the process terminates at $\mathcal{S}_i$ where $i \leq T$ and the final output is correct. We will call the special case where the language model solves the reasoning task within 0 steps of CoT as solving the reasoning task without CoT.

2.2 Representation Gap on IsTree

We study the representation gap between RNNs and Transformers on a minimal example of reasoning tasks, called IsTree: given an undirected graph G of n nodes, determine whether G is a tree, i.e., whether every pair of nodes is connected by exactly one simple path. A classical solution to IsTree is running Depth First Search (DFS), which takes $O(n)$ time.

In the context of language modeling, we can write the graph G as a sequence of tokens, and then the task of IsTree is to determine whether G is a tree by predicting a YES/NO token with or without CoT. We use the following tokenization for the graph G:

$$\text{Tokenize}(G) = \{\texttt{<s>}, u_1, \sim, v_1, u_2, \sim v_2, \ldots, u_m, \sim, v_m\}, \tag{5}$$

where <s> and $\sim$ are two special tokens representing the start of the sentence and an edge, and u_i, v_i are numbers denoting the nodes of the graph.

Our first result states that RNN with $o(n)$ bit memory cannot solve IsTree, even with an arbitrary choice of chain of thought. On the other hand, there exists a Transformer with constant size and $O(\log n)$ precision that can generate a chain-of-thought of length n following DFS and perfectly solve the same question.

Theorem 2.2. *For any n and RNN R with $o(n)$ bit memory, R cannot perfectly solve IsTree of size n, with any length of chain of thought. On the other hand, there exists a Transformer T_1 with constant dimension and depth, and $O(\log n)$ precision that can solve IsTree of size n perfectly with Chain of Thought of length $O(n)$.*

2.3 Transformer Can Simulate RNNs

The above theorem shows the existence of a task where Transformers require exponentially less memory than RNNs. However, it does not rule out the possibility that there exists a corresponding task where the Transformer will be more redundant and require exponentially more parameter than RNNs. However, the following theorem confirms that such a task doesn't exist for regular RNN (Definition A.7).

Theorem 2.3. *Given any constant $A > 0$, constant word width and number of special symbols $d, n_S > 0$, for any n, precision $p = \Theta(A \log n)$ and RNN R with word embedding $W^{(E)} \in \mathbb{R}_p^{(n+n_S)\times d}$ such that each recurrent iteration can be calculated with a circuit with size $\mathrm{P}(n) \le 2^{p/2}$, there exists a Transformer T with $O(\mathrm{P}(n) \log \max\{\mathrm{P}(n), n\})$ bit parameter and word embedding $\begin{bmatrix} W^{(E)} & \mathbf{0}^{(n+n_S)\times d} \end{bmatrix}$ that can simulate the RNN with at most n^A step chain-of-thought precisely, using at most $(\mathrm{P}(n)+1)n^A$ step chain of thought on every input with length n.*

3 The Fix: Retrieval Augmented RNNs

3.1 Definitions

In-Context Retrieval Augmentation Retrieval Augmented Generation means giving the language model the capability to retrieve relevant information to assist generation. We formally described the process here. Given a language model M with vocabulary V (containing two additional special tokens called $\mathrm{StartSearch}$ and $\mathrm{EndSearch}$) and the tokenized input sequence $\mathcal{S}_{\text{in}} \in |V|^{l_0}$, retrieval augmented generation generates following sequence of tokenized sequence:

$$\mathcal{S}_0 = \mathcal{S}_{\text{in}}, s_i^{\text{next}} = \arg\max_{j \in V} M(\mathcal{S}_i)[j],$$

$$\mathcal{S}_{i+1} = \begin{cases} \mathcal{S}_i \oplus s_i^{\text{next}}, s_i^{\text{next}} \neq \mathrm{EndSearch} \\ \mathcal{S}_i \oplus s_i^{\text{next}} \oplus \mathrm{RETRIEVE}\left(\mathcal{S}_i\right), \text{otherwise.} \end{cases}$$

Here $\mathrm{RETRIEVE}$ looks for the last occurrence of $\mathrm{StartSearch}$ at position l_s and $\mathrm{EndSearch}$ in $\mathcal{S}$ at positon l_e and treat $\mathrm{Detokenize}(\mathcal{S}_{l_s:l_e})$ as a regular expression. The algorithm then evaluate the regular expression on $\mathrm{Detokenize}(\mathcal{S}_{1:l_s-1})$ and returns the result. Here $\mathrm{Detokenize}$ maps the tokenized sequence back to the string.

Evaluating a constant length regular expression takes $\tilde{O}(n)$ time. However, because all the operation here is operated on the string, the retrieval functionality can be implemented efficiently on CPUs. Moreover, the search queries used in the following theorem Theorems 3.1 and 3.2 can be further optimized to take $\tilde{O}(1)$ amortized time.

3.2 Retrieval capability closes the representation gap

Here we would show that, by simply allowing the model to query its context, RNNs with $O(\log n)$ bit memory could be as powerful as a polynomial-time Turing machine.

Theorem 3.1. *Given any constant A, B, for any polynomial-time Turing machine $T \in \mathrm{TIME}(n^A)$ with B states and vocabulary size B, there exists a retrieval augmented RNNs (see Definition A.9) with vocabulary of B special symbol, $O(A \log n + B)$ bit memory, and $O(B^2(A \log n + B))$ parameter, that can simulate the output of T on any input with length n with a chain of thought of length $O(n^A)$.*

As a concrete example, we would show that the retrieval augmented RNNs can solve the IsTree problem with a chain of thought of length $O(n)$.

Theorem 3.2. *There exists a retrieval augmented RNNs with $O(\log n)$ bit memory and $O(\log n)$ parameter, that can solve IsTree of size n with a chain of thought of length $O(n)$.*

References

Anonymous. Chain of thought empowers transformers to solve inherently serial problems. In *Submitted to The Twelfth International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=3EWTEy9MTM`. under review.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*, 2023.

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

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b. URL `https://openreview.net/forum?id=qHrADgAdYu`.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

Zhong Li, Jiequn Han, Weinan E, and Qianxiao Li. On the curse of memory in recurrent neural networks: Approximation and optimization analysis. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=8Sqhl-nF50`.

Zhong Li, Haotian Jiang, and Qianxiao Li. On the approximation properties of recurrent encoder-decoder architectures. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=xDIvIqQ3DXD`.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=De4FYqjFueZ`.

William Merrill and Ashish Sabharwal. The Parallelism Tradeoff: Limitations of Log-Precision Transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 06 2023. ISSN 2307-387X. doi: 10.1162/tacl_a_00562. URL `https://doi.org/10.1162/tacl_a_00562`.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Jiaju Lin, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartlomiej Koptyra, Hayden Lau, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Bolun Wang, Johan S. Wind, Stanislaw Wozniak, Ruichong Zhang, Zhenyuan Zhang, Qihang Zhao, Peng Zhou, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. Rwkv: Reinventing rnns for the transformer era, 2023.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2023.

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. Llama 2: Open foundation and fine-tuned chat models, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

A Notations and Definitions

We will first define the necessary definitions and notations.

A.1 Reasoning Tasks on Graphs.

The reasoning tasks we consider in this paper are all defined on graphs. Without otherwise specified, we will use n as the number of vertices and m as the number of edges. Without loss of generality, we will assume the vertices are labeled by $[n]$.

We will focus on decision problems on graphs, which are defined as follows:

Definition A.1 (Decision Problem on Graphs)**.** A decision problem on graph is a function $f : \mathcal{G} \to \{\text{YES}, \text{NO}\}$, where $\mathcal{G}$ is the set of all possible graphs.

We will use the following decision problem as our main example:

Definition A.2 (IsTree)**.** $\text{IsTree}(G) = \text{YES}$ if G is a tree, and $\text{IsTree}(G) = \text{NO}$ otherwise.

One can view IsTree as a minimal example of reasoning tasks. One of the classical solutions to IsTree is running Depth First Search and this algorithm takes $O(n)$ time.

A.2 Numerical Precision.

We would consider computation models with fixed numerical precision in this paper. We will use p to denote the precision of the number of bits to represent real numbers. We will use $\mathbb{R}_p$ to denote the set of all real numbers that can be represented by p-bit floating point numbers. We will use $\text{ROUND}(x, p)$ to denote the rounding function that rounds x to the nearest number in $\mathbb{R}_p$. We will assume p is an odd number without loss of generality.

$$\mathbb{R}_p = \left\{(2b_p - 1)\left(\sum_{i=1}^{p-1} b_i 2^{(p-1)/2-i}\right) : \forall i \in [p], b_i \in \{0,1\}\right\}. \tag{6}$$

For calculation over $\mathbb{R}_p$, we will assume the calculation is exact and the result is rounded to $\mathbb{R}_p$ at the end, that is, for operator $\oplus$, we will have

$$\begin{aligned}&\text{ROUND}(x,p) \oplus_p \text{ROUND}(y,p)\\=&\text{ROUND}\left(\text{ROUND}(x,p) \oplus \text{ROUND}(y,p), p\right).\end{aligned}$$

We will additionally define $\mathbb{Z}_p$ as the set of all integers that can be represented by p-bit floating point numbers. We will define $1/[m]$ as the set of unit fractional $\{\frac{1}{i}\}_{i\in[m]}$. Further, we will define $\text{ROUND}(1/[m], p)$ as the rounding of $1/[m]$ to $\mathbb{R}_p$. We will additionally define for any real number $x \in \{0, 1\}$, $\text{next}(x) = \frac{1}{m+1}$ where $m = \arg\min_{k\in\mathbb{Z}} |x - \frac{1}{k}|$.

We will assume $p = O(\log n)$ in this paper and state the constant explicitly when necessary.

A.3 Models

Tokenization. To tokenize a graph G, we will order its edges $\text{E} = \{(u_i, v_i) \mid u_i < v_i\}$ randomly and tokenize it into the following string:

$$\text{Tokenize}(G) = \{\texttt{<s>}, u_1, \sim, v_1, \ldots, u_m, \sim, v_m\}. \tag{7}$$

We hereby assume there are constant number of more special tokens that are not the same as any number token, which are listed below:

- $\texttt{<s>}$: the first special token, indicating the start of a sentence.
- $\sim$: the second special token, indicating an edge.
- YES: the third special token, indicating the answer is yes.
- NO: the fourth special token, indicating the answer is no.

- $\mathrm{StartSearch}$: the fifth special token, indicating the start of a search query.
- $\mathrm{EndSearch}$: the sixth special token, indicating the end of a search query.

We will denote the total number of special tokens as n_S and the total vocabulary size as $|V| = n + n_S$. We will further define the detokenization function $\mathrm{Detokenize}$,

$$\mathrm{Detokenize}(\mathcal{S}) = \text{``}\mathcal{S}_1\ \mathcal{S}_2\ \ldots\ \mathcal{S}_l\text{''}.$$

Here each $\mathcal{S}_i$ is either a number or a special token, which we will treat as a character.

Embedding Functions. We will use d to denote the dimension of the embedding and w_i to denote the i-th standard basis vector in $\mathbb{R}^d$.

We will separate the embedding function into two parts: the word embedding and the position embedding. For the word embedding, we will use $iw_1 + i^2 w_2 \in \mathbb{R}^d$ to represent the embedding of the vertice i in the tokenization of the graph. For the k-th special token, we will use w_{2+k} to represent its embedding. For example, the embedding of $\sim$ is w_2. We will denote the word embedding matrix as $W^{(E)} \in \mathbb{R}^{|V| \times d}$.

For the position embedding, we will use lw_d to represent the position embedding of the l-th token in the tokenization of the graph, which is a hyperparameter. The final embedding of any token sequence is the sum of the word embedding and the position embedding. We will use Emb to denote the embedding function.

This embedding function will be fixed and shared across all models we consider in this paper and will not be learned during training, hence we will not consider it as part of the model parameters.

Language Modeling. In this work, we will consider the difference between Transformer and Recurrent Neural Networks on reasoning task, which is a special case of language modelling. We will define language modelling as follows:

Definition A.3 (Language Model)**.** A language model is a function $M : \cup_{l=1}^{L} |V|^l \to \mathbb{P}_{|V|}$, where $|V|$ is the vocabulary size, l is the sequence length, and $\mathbb{P}_{|V|}$ is the probability simplex over $|V|$.

We will now formally define the two types of language models we consider in this paper.

Transformers.

Definition A.4 (Transformer Block). Let $X \in \mathbb{R}^{d \times l}$ be the input matrix, where l is the sequence length. The output of a transformer block f is defined as:

$$\begin{aligned} f(X) &= X + \mathcal{A}(X) + g(X + \mathcal{A}(X)), \\ \mathcal{A}(X) &= \sum_{h=1}^{H} W^{(V,h)} X \sigma\left(\frac{\left(W^{(K,h)}X\right)^\top W^{(Q,h)} X}{\sqrt{d}} + C\right), \end{aligned} \tag{8}$$

where g is a column-wise ReLU feed-forward network with width w and output dimension d, $\mathcal{A}$ is the scaled dot-product attention, σ is the column-wise softmax function, $W^{(K,h)}$, $W^{(Q,h)}$, $W^{(V,h)}$ are the learnable parameters and H is the number of heads, and $C = \begin{bmatrix} 0 & 0 & \ldots & 0 \\ -\infty & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ -\infty & -\infty & \ldots & 0 \end{bmatrix} \in \mathbb{R}^{l \times l}$ is a mask to prevent the attention from attending to future tokens.

Definition A.5 (Transformer)**.** Let $\mathcal{S} \in |V|^l$ be the tokenized input sequence, the output of a transformer is defined as:

$$T(\mathcal{S}) = \sigma\left(W^{(E)}\left(f_L\left(\ldots f_1\left(\mathrm{Emb}\left(\mathcal{S}\right)\right)\right)\right)\right)_{:,l}. \tag{9}$$

where σ is the column-wise softmax function, f_i is the i-th transformer block. We will call the i-th transformer block the i-th layer of the transformer and denote its feed-forward layer and attention layer as g_i and $\mathcal{A}_i$ respectively.

Recurrent Neural Networks Recently there has been a lot of interest in the linear-time transformer, which replaces the full-attention calculation with linear-time alternatives. These variants are mostly special forms of recurrent neural networks that are parallelizable. We will define a general form of recurrent neural networks containing all of these variants; hence, our analysis will apply to all of them.

Definition A.6 (RNN)**.** A recurrent neural network is characterized by two functions: state transition function $\mathbf{t} : \mathbb{R}_p^\Lambda \times \mathbb{R}_p^d \to \mathbb{R}_p^\Lambda$ and output function $\mathbf{o} : \mathbb{R}_p^\Lambda \to \mathbb{R}_p^d$, where d is the dimension of the state. Let $\mathcal{S} \in |V|^l$ be the input sequence, the output of a recurrent neural network is defined as:

$$R(\mathcal{S}) = \sigma\left(W^{(E)}\mathbf{o}(s_n)\right)_{:,l}, \tag{10}$$

$$\forall k \in [n], s_k = \mathbf{t}(s_{k-1}, \text{Emb}(\mathcal{S})_{:,k}), \tag{11}$$

where $s_0 \in \mathbb{R}_p^\Lambda$ is a constant vector.

We can characterize the complexity of an RNN using the following three measures,

1. Parameter size P: the number of bit of parameters in $\mathbf{t}$ and $\mathbf{o}$.
2. State memory size M: the number of bit in memory requires to simulate a fixed parameter RNN, which in this case is $\Lambda \times p$.
3. Circuit size C: the number of bit-wise arithmetic operations needed to calculate $\mathbf{t}$ and $\mathbf{o}$.

We will further constrain ourselves to RNNs with the following properties, which still contains all the modern variants of linear-time transformers to the best of our knowledge.

Definition A.7. A recurrent neural network (Definition A.6) is regular if $\mathrm{P} = \Omega(\mathrm{M})$ and $\mathrm{C} = \tilde{\Theta}(\mathrm{P})$.

A.4 Language Models for Reasoning.

Chain of Thoughts. We will now define how do we use language models to solve reasoning tasks utilizing the following technique called chain of thought.

Definition A.8 (Chain of Thought)**.** Given a language model M with vocabulary V and the tokenized input sequence $\mathcal{S}_{\text{in}} \in |V|^{l_0}$, chain of thought (CoT) generates following sequence of tokenized sequence:

$$\mathcal{S}_0 = \mathcal{S}_{\text{in}}, \tag{12}$$

$$s_i^{\text{next}} = \arg\max_{j \in V} M(\mathcal{S}_i)[j], \tag{13}$$

$$\mathcal{S}_{i+1} = \mathcal{S}_i \oplus s_i^{\text{next}}, \forall i \geq 0. \tag{14}$$

The process terminates at $\mathcal{S}_i$ when $\arg\max_{j \in V} M(\mathcal{S}_i)[j]$ is YES or NO. The language model can solve the reasoning task within T steps of CoT if the process terminates at $\mathcal{S}_i$ where $i \leq T$ and the final output is correct. We will call the special case where the language model solves the reasoning task within 0 steps of CoT as solving the reasoning task without CoT.

Retrieval Augmentation. We will show in this paper that retrieval augmentation is a necessary technique to solve reasoning tasks for recurrent neural networks. We will define retrieval augmentation as follows:

Definition A.9 (Retrieval Augmented Generation)**.** Retrieval Augmented Generation means giving the language model the capability to retrieve relevant information to assist generation. We formally described the process here. Given a language model M with vocabulary V (containing two additional special tokens called $\mathrm{StartSearch}$ and $\mathrm{EndSearch}$) and the tokenized input sequence $\mathcal{S}_{\text{in}} \in |V|^{l_0}$, retrieval augmented generation generates following sequence of tokenized sequence:

$$\mathcal{S}_0 = \mathcal{S}_{\text{in}},$$

$$s_i^{\text{next}} = \arg\max_{j \in V} M(\mathcal{S}_i)[j],$$

$$\mathcal{S}_{i+1} = \begin{cases} \mathcal{S}_i \oplus s_i^{\text{next}}, s_i^{\text{next}} \neq \mathrm{EndSearch} \\ \mathcal{S}_i \oplus s_i^{\text{next}} \oplus \mathrm{RETRIEVE}\left(\mathcal{S}_i\right), \text{otherwise.} \end{cases}$$

Here $\mathrm{RETRIEVE}$ looks for the last occurrence of $\mathrm{StartSearch}$ at position l_s and $\mathrm{EndSearch}$ in $\mathcal{S}$ at positon l_e and treat $\mathrm{Detokenize}(\mathcal{S}_{l_s:l_e})$ as a regular expression. The algorithm then use the regular expression on $\mathrm{Detokenize}(\mathcal{S}_{1:l_s-1})$. If the regular expression ever matches, the $\mathrm{RETRIEVE}$ will return the match. If the regular expression never matches, $\mathrm{RETRIEVE}$ will return special token FAILED.

Similar to Definition A.8, we can define the notion of solving the reasoning task within T steps of retrieval augmented generation and solving the reasoning task without retrieval augmented generation.

We will note that assuming $|V| = O(n)$ and every search query and the result is of length $O(1)$, the regular expression evaluation can typically be evaluated in $O(n)$ time.

B Omitted Proof

B.1 Building Blocks of Transformers Construction

We will show in this section some construction for basic functionality using Transformer Blocks. This construction will be used in the following sections to prove the main results.

We will always use $X \in \mathbb{R}_p^{d\times l}$ as the input to the Transformer Block, where d is the dimension of the input, and l is the length of the sequence. We will first outline all the building functionality and then show how to implement them.

Definition B.1 (Copying Function). For integer s, index set $I_1, I_2 \subset [d-20]$ satisfying $|I_1| = |I_2|$, a copying function $\mathrm{COPY}[s, I_1, I_2]$ satisfies the following, $\forall X \in \mathbb{R}_p^{d\times l}$, then

$$\mathrm{COPY}[s, I_1, I_2](X)_{I_2,k} = x_{I_1,\max\{k-s,0\}} \quad \forall k \le [m]$$
$$\mathrm{COPY}[s, I_1, I_2](X)_{I_2^c,k} = 0 \quad \forall r \in [m]$$

Definition B.2 (Counting Function). For index set $I_1, I_2 \subset [d-20], |I_1| = |I_2| \le 10$ and index i, a counting function $\mathrm{COUNT}[I_1, I_2, i]$ satisfies the following, if $\forall v \in I_1 \cup I_2, k \in [l], X_{v,k} \in \mathbb{Z}_p$ and $X_{v,k} \neq 0$, then

$$\mathrm{COUNT}[I_1, I_2, i](X)_{i,k} = \frac{1}{\sum_{h=1}^{k} \mathbf{1}[X_{I_1,h} = X_{I_2,k}] + 1} \quad \forall k \in [l].$$
$$\mathrm{COUNT}[I_1, I_2, i](X)_{i^c,k} = 0 \quad \forall k \in [l].$$

Definition B.3 (Matching Function). For index set $I_1, I_2, I_3, I_4 \subset [d-20], |I_1| = |I_2| \le 10, |I_3| = |I_4|$, a matching function $\mathrm{Match}[I_1, I_2, I_3, I_4]$ satisfies the following, if $\forall v \in I_1 \cup I_2, k \in [l], X_{v,k} \in \mathbb{Z}_p$, then

$$\mathrm{Match}[I_1, I_2, I_3, I_4](x)_{I_3,k} = X_{I_4,k^*} \quad \forall k \in [l]$$
$$\text{where } k^* = \begin{cases} \min\{h \mid X_{I_1,h} = X_{I_2,k}\}, \{h \mid X_{I_1,h} = X_{I_2,k}\} \neq \emptyset \\ 1, \text{otherwise} \end{cases}.$$

Definition B.4 (Matching Next Function). Given any interger constant A, assuming $p > 10A\log n$, for index set $I_1, I_2, I_3, I_4 \subset [d-20], |I_1| = |I_2| \le 10, |I_3| = |I_4|$, and a special counting index a, a matching next function $\mathrm{MatchNext}[I_1, I_2, I_3, I_4, a]$ satisfies the following, if X satisfies the following condition:

1. $\forall v \in I_1 \cup I_2, k \in [l], X_{v,k} \in \mathbb{Z}_p$,
2. $X_{a,k} \in \mathrm{ROUND}(1/[n^A], p) \cup \{0\}$,
3. For any $k \in [l]$, given any $k \ge k$, the counting index multiset $S_k = \{X_{a,k'} \mid X_{I_1,k'} = X_{I_2,k}\}$ takes consecutive and disjoint values in $\mathrm{ROUND}(1/[n^A], p)$, that is, there exists $u_k, v_k \in \mathrm{ROUND}(1/[n^A], p)$ such that $S_k = [u_k, v_k] \cap \mathrm{ROUND}(1/[n^A], p)$.

then, we have

$$\mathrm{MatchNext}[I_1, I_2, I_3, I_4, a](X)_{I_3,k} = X_{I_4,k^*} \quad \forall k \in [l]$$
$$\text{where } k^* = \arg\min_{h \in \{h \mid X_{I_1,h} = X_{I_2,k}\} \cup \{1\}} |X_{a,h} - \mathrm{next}(X_{a,k})|.$$

Now we will show how to implement these functions using Transformer Blocks. The construction here is motivated by the construction in Feng et al. (2023a) with some modifications.

Lemma B.5 (Copying Blocks)**.** *For integer s, index set $I_1, I_2 \subset [d-10]$ satisfying $|I_1| = |I_2|$, a copying function $\mathrm{COPY}[s, I_1, I_2]$ can be implemented with 1 feedforward block g and 1 attention block $\mathcal{A}$ with 1 attention head. Formally, when $X_{d,k} = k$, it holds that*

$$\mathcal{A}\left(g\left(X\right) + X\right) = \mathrm{COPY}[s, I_1, I_2](X).$$

Proof of Lemma B.5. We will use the feedforward block to calculate $X_{k,d}^2$ and 1 and have

$$\begin{aligned}(g(X) + X)_{d-1,k} &= k^2 \\ (g(X) + X)_{d-2,k} &= 1. \\ \forall i \notin \{d-1, d-2\}, (g(X) + X)_{i,k} &= X_{i,k}.\end{aligned}$$

We will use X' to denote $g(X) + X$. Then we will choose $W^{(K)}, W^{(Q)}$ such that

$$\begin{aligned}W^{(K)} X'_{:,k'} &= n \begin{bmatrix} 1 \\ k' \\ k'^2 \end{bmatrix} \\ W^{(Q)} X'_{:,k} &= \begin{bmatrix} -(k^2 + s^2 - 2sk) \\ 2k - 2s \\ -1 \end{bmatrix}\end{aligned}$$

Hence

$$\begin{aligned}&\left(\left(W^{(K)} X\right)^\top \left(W^{(Q)} X\right)\right)_{k',k} \\ = & -n\left(k'^2 - k'(2k - 2s) + k^2 + s^2 - 2sk\right) \\ = & -n(k - s - k')^2\end{aligned}$$

Hence we have

$$\arg\max_{k' < k} \left(\left(W^{(K)} X\right)^\top \left(W^{(Q)} X\right)\right)_{k',k} = \max\{k - s, 0\}.$$

Also, for any $k' \leq k, k' \neq \max\{k - s, 0\}$, we have

$$\left(\left(W^{(K)} X\right)^\top \left(W^{(Q)} X\right)\right)_{k',k} - \left(\left(W^{(K)} X\right)^\top \left(W^{(Q)} X\right)\right)_{\max\{k-s,0\},k} < -n.$$

Hence after the column-wise softmax and rounding to $p = O(\log n)$ bit, we have

$$\left(\sigma\left(\left(W^{(K)} X\right)^\top \left(W^{(Q)} X\right)\right)\right)_{k',k} = \mathbf{1}[k' = \max\{k - s, 0\}].$$

We will then choose $W^{(V)}$ such that

$$\begin{aligned}W^{(V)} X'_{I_2,k'} = X'_{I_1,k'} = X_{I_1,k'} \quad &\forall k' \in [l]. \\ W^{(V)} X'_{I_2^c,k'} = 0 \quad &\forall k' \in [l].\end{aligned}$$

This then concludes that

$$\mathcal{A}\left(g\left(X\right) + X\right) = \mathrm{COPY}[s, I_1, I_2](X).$$

The proof is complete. □

Lemma B.6 (Counting Blocks). *For index set $I \subset [d-20]$ satisfying $|I_1| = |I_2| \leq 10$, a counting function $\mathrm{COUNT}[i, I_1, I_2]$ can be approximated with 1 feedforward block g and 1 attention block $\mathcal{A}$ with 1 attention head. Formally, when $X_{d,k} = k$ and $X_{3,k} = \mathbf{1}[k=1], X_{I_1,1} = 0$, it holds that*

$$\mathcal{A}\left(g\left(X\right)+X\right)_{i,k} = \mathrm{ROUND}\left(\mathrm{COUNT}[s, I_1, I_2](X)_{i,k}, p\right).$$
$$\mathcal{A}\left(g\left(X\right)+X\right)_{i^c,k} = 0.$$

Proof of Lemma B.6. We will use the feedforward block to calculate $X^2_{v,k}, v \in I_1 \cup I_2$ and have

$$\begin{aligned}
(g(X)+X)_{d-i,k} &= X^2_{I_1[i],k}, i \in [|I|].\\
(g(X)+X)_{d-|I|-i,k} &= X^2_{I_2[i],k}, i \in [|I|].\\
(g(X)+X)_{d-2|I|-1,k} &= 1.\\
\forall i \notin \{d-i \mid i \in [2|I|+1]\}, (g(X)+X)_{i,k} &= X_{i,k}.
\end{aligned}$$

We will use X' to denote $g(X)+X$. Then we will choose $W^{(K)}, W^{(Q)}$ such that

$$W^{(K)} X'_{:,k'} = n \begin{bmatrix} 1+\mathbf{1}[k'=1] \\ X_{I_1[i],k'} \\ X^2_{I_1[i],k'} \end{bmatrix}_{i\in[I]}$$
$$W^{(Q)} X'_{:,k} = \begin{bmatrix} X^2_{I_2[i],k} \\ -X_{I_2[i],k} \\ 1 \end{bmatrix}_{i\in[I]}$$

Hence,

$$\begin{aligned}
&\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k}\\
=& -n\sum_{i=1}^{|I|}\left(X'^2_{I_2[i],k'} - X_{I_1[i],k'}(2X_{I_2[i],k}) + X^2_{I_2[i],k}\right) + n\mathbf{1}[k'=1]\sum_{i=1}^{|I|} X^2_{I[i],k}.\\
=& -n\sum_{i=1}^{|I|}(X_{I_1[i],k'} - X_{I_2[i],k})^2 + n\mathbf{1}[k'=1]\sum_{i=1}^{|I|} X^2_{I_2[i],k}.
\end{aligned}$$

Hence we have

$$\max_{k'<k}\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k} = 0.$$

Equality holds when $k'=1$ or $X_{I_1[i],k'} = X_{I_2[i],k}$ for all $i \in [|I_1|]$.

Also, for any $k' \leq k, k' \neq 1$ or $X_{I_1[i],k'} \neq X_{I_2[i],k}$ for some $i \in [|I_1|]$, we have

$$\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k} < -n.$$

Hence after the column-wise softmax and rounding to $p = O(\log n)$ bit, we have

$$\left(\sigma\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)\right)_{k',k} = \mathrm{ROUND}\left(\frac{1}{\sum_{h=1}^{k}\mathbf{1}[X_{I_1,h} = X_{I_2,k}]+1}, p\right)$$

Here the $O\left(\frac{1}{n^A}\right)$ term comes from the fact that the softmax is rounded to $p = O(\log n)$ bit.

We will then choose $W^{(V)}$ such that

$$W^{(V)} X'_{i,k'} = X'_{3,k'} = \mathbf{1}[k' = 1] \quad \forall k' \in [l].$$
$$W^{(V)} X'_{I^c,k'} = 0 \quad \forall k' \in [l].$$

This then concludes that

$$\mathcal{A}\left(g\left(X\right) + X\right)_{i,k} = \text{ROUND}\left(\text{COUNT}[s, I_1, I_2](X)_{i,k}, p\right).$$
$$\mathcal{A}\left(g\left(X\right) + X\right)_{i^c,k} = 0.$$

□

Lemma B.7 (Matching Blocks)**.** *Given any constant c, for index set $I_1, I_2, I_3, I_4 \subset [d-20], |I_1| = |I_2| \leq 10, |I_3| = |I_4|$, a matching function* $\text{Match}[I_1, I_2, I_3, I_4]$ *can be implemented with* 1 *feed-forward block g and* 1 *attention block $\mathcal{A}$ with* 1 *attention head. Formally, when $X_{d,k} = k, X_{3,k} = \mathbf{1}[k = 1], X_{I_1,1} = 0$ and $k \leq n^c$, it holds that*

$$\mathcal{A}\left(g\left(X\right) + X\right) = \text{Match}[I_1, I_2, I_3, I_4](X)$$

Proof. We will use the feedforward block to calculate $k^2, X_{v,d}^2, v \in \cup I_1 \cup I_2$ as in the proof of Lemmas B.5 and B.6.

We then choose $W^{(K)}, W^{(Q)}$ such that

$$\begin{aligned}&\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k} \\ =& - n^{4c+1} \sum_{i=1}^{|I|} (X_{I_1[i],k'} - X_{I_2[i],k})^2 - nk'^2 \\ &+ \mathbf{1}[k' = 1]\left(n^{4c+1} \sum_{i=1}^{|I|} X_{I_2[i],k}^2 + n - n^{2c+2}\right).\end{aligned}$$

The detailed construction of $W^{(K)}, W^{(Q)}$ is omitted here since it is similar to the proof of Lemmas B.5 and B.6.

We will discuss several cases for the distribution of $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k}$. It always holds that $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{1,k} = -n^{2c+2}$.

1. If there doesn't exists k', such that $X_{k',I_1} = X_{k,I_2}$, then for any $i > 1$, we have $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{i,k} < -n^{4c+1}$.
2. If there exists k', such that $X_{k',I_1} = X_{k,I_2}$, then for such k', we have $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k} = -nk'^2 > -n^{2c+1}$. The rest of the entries are all smaller than $-n^{4c+1}$. Finally, the largest k' satisfying that $X_{k',I_1} = X_{k,I_2}$ will corresponds to a $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k}$ that is at least n larger than the second largest $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k}$, as in the proof of Lemma B.5.

Concluding the above discussion, we have after the column-wise softmax and rounding to $p = O(\log n)$ bit,

$$\left(\sigma\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)\right)_{k',k} = \begin{cases}\mathbf{1}[k' = \min\{h \mid X_{I_1,h} = X_{I_2,k}\}], \{h \mid X_{I_1,h} = X_{I_2,k}\} \neq \emptyset \\ \mathbf{1}[k' = 1], \text{otherwise}\end{cases}$$

Further, we will choose $W^{(V)}$ such that

$$W^{(V)} X'_{I_3,k'} = X'_{I_4,k'} = X_{I_4,k'} \quad \forall k' \in [l].$$
$$W^{(V)} X'_{I_3^c,k'} = 0 \quad \forall k' \in [l].$$

This then concludes that

$$\mathcal{A}\left(g\left(X\right) + X\right) = \mathrm{Match}[I_1, I_2, I_3, I_4](X)$$

This concludes the proof. □

Lemma B.8 (Matching Next Blocks)**.** *Given any constant A, c, for index set $I_1, I_2, I_3, I_4 \subset [d-20], |I_1| = |I_2| \le 10, |I_3| = |I_4|$ and a special counting index a, a matching next function $\mathrm{MatchNext}[I_1, I_2, I_3, I_4, a]$ can implement with 1 feedforward block g and 1 attention block $\mathcal{A}$ with 1 attention head. Formally, when $X_{d,k} = k, X_{3,k} = \mathbf{1}[k=1], X_{I_1,1} = 0$ and $k \le n^c$, it holds that*

$$\mathcal{A}\left(g\left(X\right) + X\right) = \mathrm{MatchNext}[I_1, I_2, I_3, I_4, a](X)$$

Proof. We will use the feedforward block to calculate the following $\overline{\mathrm{next}}$ function, where

$$\overline{\mathrm{next}}(x) = \begin{cases} \frac{1}{2}, & x \ge \frac{2}{3}. \\ \frac{1}{3}, & \frac{3}{5} > x > \frac{2}{5}. \\ \frac{1}{4}, & \frac{7}{20} > x > \frac{3}{10} \\ x - x^2 + x^3, & x \le \frac{11}{40}. \end{cases}$$

The value can be arbitrary for $x \in [\frac{11}{40}, \frac{3}{10}] \cup [\frac{2}{5}, \frac{7}{20}] \cup [\frac{3}{5}, \frac{2}{3}]$.

The purpose of this is to approximate the next function for $x \in \mathrm{ROUND}(1/[n^A], p)$, and we have the following lemma.

Lemma B.9. *For large enough n and any $x \in \mathrm{ROUND}(1/[n^A], p)$, we have*

$$|\overline{\mathrm{next}}(x) - \mathrm{next}(x)| \le \mathrm{next}(x)^3 + O\left(\frac{1}{n^{10A}}\right).$$

Proof. We always have $\mathrm{ROUND}(1/[n^A], p) \cap \left([\frac{11}{40}, \frac{3}{10}] \cup [\frac{2}{5}, \frac{7}{20}] \cup [\frac{3}{5}, \frac{2}{3}]\right) = \emptyset$. We will discuss several cases for $x \in \mathrm{ROUND}(1/[n^A], p)$.

1. If $x \ge \frac{3}{10}$, then $\overline{\mathrm{next}}(x) = \mathrm{next}(x)$.
2. If $x \le \frac{7}{20}$, it holds that $|x - 1/m| \le 1/n^{10A}, m \ge 3$, then

$$\begin{aligned} \overline{\mathrm{next}}(x) = x - x^2 &= \frac{1}{m} - \frac{1}{m^2} + \frac{1}{m^3} + O\left(\frac{1}{n^{10A}}\right) \\ &= \frac{1}{m+1} - \frac{1}{m^3(m+1)} + O\left(\frac{1}{n^{10A}}\right) \end{aligned}$$

This then concludes the proof. □

We then choose $W^{(K)}, W^{(Q)}$ such that

$$\begin{aligned} &\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k} \\ =& -n^{4A+3}\sum_{i=1}^{|I|}(X_{I_1[i],k'} - X_{I_2[i],k})^2 - n^{4A+1}(\overline{\mathrm{next}}(X_{a,k}) - X_{a,k'})^2 \\ &+ \mathbf{1}[k'=1]\left(n^{4A+3}\sum_{i=1}^{|I|}X_{I_2[i],k}^2 + n^{4A+1}X_{a,k}^2 - n^{4A+2}\right). \end{aligned}$$

Again, the detailed construction of $W^{(K)}, W^{(Q)}$ is omitted here since it is similar to the proof of Lemmas B.5 and B.6.

We will discuss several cases for the distribution of $\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}$. It always holds that $\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{1,k} = -n^{4A+2}$.

1. If there doesn't exists k', such that $X_{k',I_1} = X_{k,I_2}$, then for any $i > 1$, we have $\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{i,k} < -n^{4A+3}$.

2. If there exists k', such that $X_{k',I_1} = X_{k,I_2}$, then for such k', we have $\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k} = -n^{3A}(\text{next}(X_{a,k}) - X_{a,k'})^2 > -n^{4A+1}$. The rest of the entries are all smaller than $-n^{4A+2}$.

   It remains to discuss the distribution of $\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}$ for k' satisfying $X_{k',I_1} = X_{k,I_2}$. When X satisfies the condition in Definition B.4, we have that $S_k = \{X_{a,k'} \mid X_{k',I_1} = X_{k,I_2}\}$ takes consecutive and disjoint values in $\text{ROUND}(1/[n^A], p)$. Hence, if $|S_k| > 2$, suppose $y, z \in S_k$ satisfies that

$$\begin{aligned}|y - \overline{\text{next}}(X_{a,k})| &= \min_{x\in S_k} |x - \overline{\text{next}}(X_{a,k})| \\ |z - \overline{\text{next}}(X_{a,k})| &= \min_{x\in S_k, x\neq y} |x - \overline{\text{next}}(X_{a,k})|.\end{aligned}$$

   We will discuss several cases for y, z.

   - If $y - \overline{\text{next}}(X_{a,k})$ and $z - \overline{\text{next}}(X_{a,k})$ are both negative, then $y > z$, we have,

$$\begin{aligned}\left(y - \overline{\text{next}}(X_{a,k})\right)^2 - \left(z - \overline{\text{next}}(X_{a,k})\right)^2 &= (y - z)(y + z - 2\overline{\text{next}}(X_{a,k})) \\ &\leq -(y-z)^2 \leq -\frac{1}{4n^{4A}}.\end{aligned}$$

   .

   - If $y - \overline{\text{next}}(X_{a,k})$ and $z - \overline{\text{next}}(X_{a,k})$ are both positive, then $y < z$, and same as above we have

$$\begin{aligned}\left(y - \overline{\text{next}}(X_{a,k})\right)^2 - \left(z - \overline{\text{next}}(X_{a,k})\right)^2 &= (y - z)(y + z - 2\overline{\text{next}}(X_{a,k})) \\ &\leq -(y-z)^2 \leq -\frac{1}{4n^{4A}}.\end{aligned}$$

   .

   - If $y - \overline{\text{next}}(X_{a,k})$ and $z - \overline{\text{next}}(X_{a,k})$ have different signs, then according to Lemma B.9, we have, $y = \text{ROUND}(\text{next}(X_{a,k}), p)$ because S_k takes consecutive and disjoint values in $\text{ROUND}(1/[n^A], p)$. This then implies that

$$\begin{aligned}&\left(y - \overline{\text{next}}(X_{a,k})\right)^2 - \left(z - \overline{\text{next}}(X_{a,k})\right)^2 \\ \leq& O\left(\frac{1}{n^{10A}}\right) + \frac{1}{\text{next}^6(X_{a,k})} - \left(\frac{1}{\text{next}(X_{a,k})(\text{next}(X_{a,k})+1)}\right)^2 \\ \leq& -\frac{1}{4n^{4A}}.\end{aligned}$$

   Concluding, we always have for any $k'' \neq k^* = \arg\max_{k',k}\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}$

$$\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k} - \left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k^*,k} \leq -\frac{n}{4}.$$

Concluding the above discussion, we have after the column-wise softmax and rounding to $p = O(\log n)$ bit,

$$\left(\sigma\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)\right)_{k',k} = \mathbf{1}\left[k' = \arg\min_{h\in\{h|X_{I_1,h}=X_{I_2,k}\}\cup\{1\}} |X_{a,h} - \text{next}(X_{a,k})|\right].$$

Further, we will choose $W^{(V)}$ such that

$$W^{(V)}X'_{I_3,k'} = X'_{I_4,k'} = X_{I_4,k'} \quad \forall k' \in [l].$$
$$W^{(V)}X'_{I_3^c,k'} = 0 \quad \forall k' \in [l].$$

This then concludes the proof. □

B.2 Proof of Theorem 2.2

We will now proceed to prove our first theorem, which states that Transformers with chain-of-thought can solve IsTree perfectly, while RNNs cannot. We will first restate the theorem here.

Theorem 2.2. *For any n and RNN R with $o(n)$ bit memory, R cannot perfectly solve IsTree of size n, with any length of chain of thought. On the other hand, there exists a Transformer T_1 with constant dimension and depth, and $O(\log n)$ precision that can solve IsTree of size n perfectly with Chain of Thought of length $O(n)$.*

Proof of Theorem 2.2. We will prove this theorem by proving the following lemmas.

Lemma B.10. *For any n and RNN R with $o(n)$ memory, R can't perfectly solve IsTree of size n.*

Lemma B.11. *There exists a Transformer T with constant depth and width, and $O(\log n)$ precision, that can solve IsTree of size n perfectly with Chain of Thought.*

This proof is a direct combination of Lemmas B.10 and B.11. □

B.2.1 Proof of Lemma B.10

Proof of Lemma B.10. The key observation is that the recurrent form of RNNs allowed the algorithm to be run in a streaming fashion with $o(n)$ bit memory. Here streaming means that the algorithm gets to look at each bit of the memory sequentially and can only update a constant size of memory. However, any streaming algorithm for IsTree requires $\Omega(n)$ memory. This result is well known in the literature of streaming algorithms. We will demonstrate the proof of it here for completeness.

Lemma B.12. *Consider the following two-party game, where Alice receives string $x \in \{0,1\}^n$ and Bob receives an integer k, and Bob wants to know the value of x_k. If only Alice is allowed to send signal to Bob, then $\Omega(n)$ bit communication complexity is required.*

Proof of Lemma B.12. Suppose there exists a communication protocal where B only receives $o(n)$ bit and can perfectly decides x_k. Because Alice don't know k, the protocal must send the same message to Bob for all k. Hence Bob can actually reconstruct the whole string x with n bit with $o(n)$ bit communication. This is a contradiction. □

Lemma B.13. *Consider the following two-party game, where Alice receives string $x \in \{0,1\}^n$ and Bob receives an integer k, and Bob wants to know whether $x_k = x_{k-1}$. If only Alice is allowed to send information, $\Omega(n)$ bit communication complexity is required.*

Proof of Lemma B.13. We can reduce this problem to the problem in Lemma B.12. Considering the game in Lemma B.12, given any $x \in \{0,1\}^n$, we can construct $\tilde{x}_i = \sum_{j=1}^i x_i \mod 2$. Then $\tilde{x}$ is a string of length n with only 0 and 1. Moreover, $x_k = x_{k-1}$ if and only if $\tilde{x}_k = \tilde{x}_{k-1}$. Hence, if Bob can solve the problem in Lemma B.13 with $o(n)$ bit, he can solve the problem in Lemma B.12. This is a contradiction. □

Now suppose that we have a streaming algorithm for IsTree with only $o(n)$ memory. We shall prove Alice and Bob in Lemma B.13 can use it to solve the original question with $o(n)$ memory.

Consider the following graph with $n+2$ nodes. There is a node i corresponding to each x_i for $i \in [n]$ and two special nodes $n+1, n+2$. Node i will be connected to $n+1$ if $x_i = 0$ and to $n+2$ if $x_i = 1$. Moreover, $k-1$ and k will be connected. Now the original answer Bob wants is False if and only if the graph is a Tree. Hence, given access to the streaming algorithm, Alice can run it on the edges that she knows exist and send the memory to Bob. Bob can then run it on the edges that he knows exist. Combining they will be able to solve the original problem. This is a contradiction.

Moreover, as RNN with chain-of-thought is also a streaming algorithm, it also requires $\Omega(n)$ memory. □

B.2.2 Proof of Lemma B.11

Proof of Lemma B.11. The proof is two-folded. We will first define an algorithm that can solve IsTree by generating a sequence of vertices of length $O(n)$, and then we will show that this sequence can be generated by a Transformer with constant depth and width, and $O(\log n)$ precision as a Chain of Thought.

Algorithm 1 Depth-First Search Algorithm

```
Require: A graph $G = (V, E)$ with $n$ vertices and $E$ has an ordering $e_1, \ldots, e_m$.
1: Initialize two stacks of vertices $S_1, S_2$ with $S_1 = [v_1], S_2 = \emptyset$.
2: while $S_1$ is not empty do
3:     Let $v$ be the top of $S_1$. Yield $v$.
4:     if there exists a neighbor $u$ of $v$ not in $S_1 \cup S_2$ then
5:         Choose $u$ such that edge $(u, v)$ has the smallest possible order and push $u$ to $S_1$.
6:     else
7:         Pop $v$ from $S_1$ and push $v$ to $S_2$.
8:     end if
9: end while
```

Algorithm for IsTree. We define Algorithm 1 as a depth-first search algorithm that can generate a sequence of vertices of length $O(n)$ that can be used to solve IsTree. We will use two stacks S_1, S_2 to store the vertices. S_1 will be used to store the vertices that are not yet visited, and S_2 will be used to store the vertices that are already visited. The algorithm will start with $S_1 = [v_1]$ and $S_2 = \emptyset$. At each step, the algorithm will pop the top of S_1 and push it to S_2. Then it will push all the neighbors of the popped vertex that are not in $S_1 \cup S_2$ to S_1. The algorithm will terminate when S_1 is empty. We will denote the yielded vertice sequence for G as $\mathcal{A}(G)$. The following lemma shows the connection between the result of the algorithm and the IsTree problem.

Lemma B.14. *For any graph G, $\mathcal{A}(G)$ is a tree-traversal of a spanning tree of the connected component of G containing v_1. Hence* IsTree(G) *is True if and only if G has $n-1$ edges and $\mathcal{A}(G)$ contains $2n-1$ vertices.*

Proof of Lemma B.14. First, every vertices in the connected component of G containing v_1 will be visited. This is because the algorithm will always push all the neighbors of the popped vertex that are not in $S_1 \cup S_2$ to S_1. Hence, the algorithm will terminate when all the vertices in the connected component of G containing v_1 are visited.

Second, every two consecutive vertices in the yielded sequence will be connected by an edge. This is because the algorithm will always push one of the neighbors of the popped vertex that are not in $S_1 \cup S_2$ to S_1. Hence, every two consecutive vertices in the yielded sequence will be connected by an edge. On the other hand, the combination of these edges will form a tree because the algorithm will never push a vertex that is already in $S_1 \cup S_2$ to S_1. Hence, the yielded sequence is a tree-traversal of a spanning tree of the connected component of G containing v_1. □

Construction of Transformer. We will now show that the yielded sequence of Algorithm 1 can be generated by a Transformer with constant depth and width, and $O(\log n)$ precision as a Chain of Thought. The Transformer will generate a valid yielded sequence but can terminate early if the graph is not a tree. We will now describe the Transformer in detail. We will assume the input token sequence $\mathcal{S}$ is as followed,

$$\mathcal{S} = \text{Tokenize}(G), v_1, \ldots v_r \tag{15}$$

for some $r \geq 0$ and $v_1 \ldots v_r$ is a valid yielded sequence. The length of $\text{Tokenize}(G)$ is $3n - 2$ with 3 tokens for each edges and 1 special token <s>. We will further denote the input to first layer X as $\text{Emb}(\mathcal{S})$. We will similarly denote the input to layer ℓ as $X^{(\ell)}$. We will also denote the output of last layer as X^{out}.

1. **Layer 1 and Layer 2 Attention.** The attention at Layer 1 will output zero and the FFN at Layer 1 and Attention at Layer 2 will implement a counting function (Definition B.2) to count the number of vertices n appears in the previous token sequence and write $\text{ROUND}\left(\frac{1}{n}, p\right)$ in a new dimension i_1 as a result.

2. **Layer 2 FFN and Layer 3 Attention.** The FFN at Layer 2 and Attention at Layer 3 will implement a copying function (Definition B.1) copying the first dimension and the counting dimension i_1 of each token to its successor at two new dimension i_2 and i_3 . For each edge, this moves type of the first vertice and the number of time the first vertice appears to $\sim$. For every vertice in the chain of thought, this moves type of the previous vertice to them.

3. **Layer 3 FFN and Layer 4 Attention.** The FFN at Layer 3 and Attention at Layer 4 will implement another copying function, copying the dimension i_2 and i_3 of each token to its successor at two new dimension i_4 and i_5. Especially, for each edge, this moves type of the first vertice and the number of time the first vertice appears to the position corresponding to the second vertices.

4. **Layer 4 FFN.** This FFN will process the information gathered from previous layer and prepare for the next layer. It will make sure the following properties hold for $X^{(5)}$,

   - For every token, the position number, its square and 1 will be kept in the last three dimension.
   - For the first vertices in each edges, $\sim$ and <s> The rest dimension will be zero.
   - For the second vertices of each edges (a, b), there will be four dimensions i_6, i_7, i_8, i_9 with value a, b and $n_{a,e}, n_{b,e}$, where $n_{a,e} = \text{ROUND}(\frac{1}{1+\#a\text{appears up to current edge}}, 1)$.
   - For vertice v_l in $v_1, \ldots, v_r$, there will be four dimensions $i_{10}, i_{11}, i_{12}, i_{13}$ with value v_l, v_{l-1} and v_l^2, v_{l-1}^2 ($v_0 = 0$).

5. **Layer 5 Attention.** Combining with the previous Layer 4 FFN layer, we will implement two match functions with two attention heads matching (i_{10}, i_{11}) or (i_{11}, i_{10}) with (i_6, i_7) at Layer 5 Attention, i.e. finding the edge in input for each step in chain of thought, we will then copy $n_{v_l,(v_l,v_{l-1})}$ to dimensions i_8 and i_9.

6. **Layer 6.** We will use Layer 5 FFN and Layer 6 Attention to implement the match function that match dimension i_{10} of current token to i_{10} in previous token. This will match v_l to the first appearance of v_l in chain of thought and we will copy i_{11} of the matched token to i_{22}. This dimension will be the first predecessor of v_l in the chain of thought (0 for v_1). We will denote this predecessor of v_l to be $f(v_l)$ as it is the father of v_l in the tree. Now we will need to split into two cases depending on whether v_{l-1} is $f(v_l)$. If $v_{l-1} = f(v_l)$ or $v_{l-1} = 0$ (for v_1), we will set dimension i_8 to be 1 and i_9 to be 0. Otherwise, we will keep dimension i_8 and i_9 as $n_{v_l,(v_l,v_{l-1})}$.

7. **Layer 7.** Now we will use Layer 6 FFN and Layer 7 Attention with two attention heads to implement two match next functions[1] (Definition B.4) which use i_8 or i_9 as the counting index, and match v_l at i_{10} to i_6 or i_7 respectively. We will then copy dimensions i_6 to i_9 of the matched tokens to i_{14} to i_{21} (because there will be two of them).

   The match next function will be able to retrieve the first edge containing v_1. For any $i \geq 2$, one of the matches next function will be able to retrieve the next edge containing v_i after (v_i, v_{i+1}) if it exists. If it doesn't exist, the corresponding counting dimension will either be zero or no smaller than $n_{v_l,(v_l,v_{l-1})}$. We will use Layer 6 FFN to decide whether the next edge exists and set dimension i_{14} of the output of Layer 6 to be the other edge in the next edge if it exists, or 0

[1]The constant A here in Definition B.4is 1

otherwise and i_{15}, i_{16} of the output of layer 6 to be the counting dimension of the next edge if it exists, or 0 otherwise. For each edges in the original input, we will also set dimension i_{15}, i_{16} to be the counting dimension of the edge.

8. **Layer 8 Attention** We will grab the next edge again, in the same manner as Layer 6, but this time using dimension i_{15} and i_{16}. The necessity of this step is that the next edge containing (v_{i-1}, v_i) in the original graph can be the same as the $(f(v_l), v_i)$ and in such case we need to check whether the next edge after this edge.

9. **Layer 8 FFN.** We now have, at each position corresponding to v_l, the first edge $(f(v_l), v_l)$ in the yielded sequence containing v_l and the other vertex in the edge containing v_l that hasn't been visited if it exists. If they don't exist, the corresponding dimension will be zero. This allow us to use Layer 8s FFN to decide the next vertex in the yielded sequence, which is exactly the first vertex different with $f(v_l)$ in the two edges if they exist, or $f(v_l)$ otherwise. We will use Layer 8 FFN to calculate the potential next vertex and put it in dimension i_{23} and its square in i_{24}.

10. **Layer 9 Attention.** Combining with Layer 8 FFN, we will match i_{23} of the current token to i_{10} of the previous token to find the first appearance of the potential next vertex in the chain of thought. We will then copy dimension d of the matched token to i_{25}. This value being 1 will imply this vertex has never appeared in the chain of thought before and any other value will imply the vertex has appeared before.

11. **Layer 9 FFN.** We can now check several cases,

    - If the potential next vertex v is either $f(v_r) \neq 0$ or never appears in the chain-of-thought sequence, then Layer 9 will output $n[-v, 1, -1, \ldots - 1]$, which will decodes to v.
    - If the potential next vertex v is not $f(v_r)$ and appears in the chain-of-thought sequence, then Layer 9 will output nw_6, which will decodes to NO, because the chain of thought has already visited v and hence the graph is not a tree.
    - If $v_r = 1$ and the potential next vertex v is $f(v_r) = 0$, this means the chain of thought has finished. In this case, layer 9 FFN will check whether the position is $3n-2+2n-1 = 5n-3$ and output nw_5 if it is, or output nw_6 otherwise, which will decodes to YES and NO respectively.

This concludes the construction of the Transformer. □

B.3 Proof of Theorem 2.3

We will now prove Theorem 2.3. We will first restate the theorem for convenience.

Theorem 2.3. *Given any constant $A > 0$, constant word width and number of special symbols $d, n_S > 0$, for any n, precision $p = \Theta(A \log n)$ and RNN R with word embedding $W^{(E)} \in \mathbb{R}_p^{(n+n_S)\times d}$ such that each recurrent iteration can be calculated with a circuit with size $\mathrm{P}(n) \leq 2^{p/2}$, there exists a Transformer T with $O(\mathrm{P}(n) \log \max\{\mathrm{P}(n), n\})$ bit parameter and word embedding $\begin{bmatrix} W^{(E)} & \mathbf{0}^{(n+n_S)\times d}\end{bmatrix}$ that can simulate the RNN with at most n^A step chain-of-thought precisely, using at most $(\mathrm{P}(n)+1)n^A$ step chain of thought on every input with length n.*

Proof. The proof is motivated by the Theorem B.15 from Anonymous (2023).

Theorem B.15 (Theorem 3.3 of Anonymous (2023))**.** *For any n and any circuit with size $T(n)$ and input size n, there exists a Transformer with constant depth and precision, $O(\log n)$ width, and a position embedding with size $O(T(n)\log n)$, such that for any input $\mathcal{S}$ of length n, the Transformer computes the output of the circuit on $\mathcal{S}$ using $T(n)$ steps.*

However, direct utilization of Theorem B.15 is not feasible because we are interested in (1) $O(\log n)$ precision Transformer, and (2) simulating the RNN for n^A step, which would corresponding to a circuit with $n^A \mathrm{P}(n)$ in size. However, as the calculation is recurrent, we can encode the RNN circuit in $O(\mathrm{P}(n))$ parameter instead.

To do so, we will unroll the circuit of each recurrent step of the RNN into $\mathrm{P}(n)$ gates. We will then assign each gate a unique id in $[\mathrm{P}(n)]$ and assume the circuit is calculated in the order of the gate id in the following manner.

1. Each gate has a type $t(i)$, which is either a constant gate outputting 1, an input gate, a hiddenstate gate, an AND gate or a XOR gate.
2. Each gate i's output depends on two values $l(i)$ and $r(i)$. If $t(i)$ is a constant gate, then $l(i)$ and $r(i)$ are assigned to be 0. When it is an input gate, $l(i)$ will be assigned to be the coordinate of the input embedding, and $r(i)$ will be assigned to be the index of the bit of the value at $l(i)$ coordinate. When it is a hiddenstate gate, $l(i)$ will be assigned to be the coordinate of the hiddenstate embedding, and $r(i)$ will be assigned to be the index of the bit of the value at $l(i)$ coordinate. If it is an AND gate or a XOR gate, $l(i)$ and $r(i)$ will be assigned to be the id of the two gates that it depends on.

We will further assume without loss of generality that the hiddenstate gate is the first $p\Lambda$ gate. The output of the last $p\Lambda$ gate will be the next hiddenstate. We will also assume that the last $p(\Lambda + d)$ to $p\Lambda - 1$ gates are the output gates. We will now first describe the chain of thought that the Transformer will output and then construct the Transformer.

Chain of thought Taking any input $\mathcal{S}$ with length n, the transformer will output a sequence of 0 and 1 tokens. The first n tokens will be the same as the input sequence. For each $a \geq 0$ and $b \in [\mathrm{P}(n) + 1]$, the $n + a\,(\mathrm{P}(n) + 1) + b$ token is

1. the output of gate b when RNN circuit is calculating the output at a position plus 1, if $b \leq \mathrm{P}(n)$.
2. the $n + a + 1$ token in the RNN chain of thought, if $b = \mathrm{P}(n) + 1$.

Construction of the Transformer.

1. **Layer 1.** The first attention layer will output zero and the first FFN layer will be of width $O(\mathrm{P}(n))$, encoding all the gate information. The output of the first layer at position $n + a\,(\mathrm{P}(n) + 1) + b$ will have the following coordinate:
   - The input i will be encoded in the first dimensions.
   - a, a^2, b, b^2 will be encoded in four different dimensions.
   - The gate type $t(s(b))$ will be encoded in the next dimension, where $s(b) = (b + 1) \mod (\mathrm{P}(n) + 1)$ If $b = \mathrm{P}(n) - 1$, then the gate type will be encoded as 0.
   - The necessary dependence $l(s(b)), l^2(s(b))$ and $r(s(b)), r^2(s(b))$ will be encoded in the next two dimensions.
   - A constant 1 will be encoded in the next dimension.
2. **Layer 2 Attention.** Together with the Layer 1 FFN, the Layer 2 Attention will implement two match functions (Definition B.3) to copy the output of gate $l(b + 1)$ and $r(b + 1)$ when RNN circuit is calculating the output at a position. When the type of gate $b + 1$ is not AND or XOR, the result will not be used in following calculation.
3. **Layer 2 FFN** Layer 2 FFN will be of width $O(1)$. The output of the layer will be
   - When $b < \mathrm{P}(n)$ and $t(s(b))$ is AND or XOR or constant, one dimension of the output will be the output of gate $b + 1$ when RNN circuit is calculating the output at a position.
   - When $b < \mathrm{P}(n)$ and $t(s(b))$ is a input or hiddenstate gate or $b = \mathrm{P}(n) + 1$, two dimensions of the output will be the position in the current chain of thought where the input bit or hiddenstate bit is copied from and its square.
   - When $b = \mathrm{P}(n)$, the output remains the same as the input to Layer 3 FFN.

4. **Layer 3 Attention.** Layer 3 Attention will be of width $O(1)$. Together with Layer 2 FFN, the Layer 3 Attention will implement match functions (Definition B.3) to copy the output at position where the input bit or hiddenstate bit is copied from. When the type of gate $b+1$ is not input or hiddenstate gate, the result will not be used in following calculation.

5. **Layer 3 FFN** Layer 3 FFN will be of width $O(1)$. The output of the layer will be
   - When $b \neq \mathrm{P}(n)$, one dimension of the output will be output of gate $s(b)$ when RNN circuit is calculating the output at $a + \mathbf{1}[b = \mathrm{P}(n) + 1]$ position.
   - When $b = \mathrm{P}(n)$, the output remains the same as the input to Layer 4 FFN.

6. **Layer 4** Layer 4 Attention will have $p-1$ heads and each head will be of width $O(1)$. Head $h \in [p-1]$ will copy the first dimension of the output of Layer 3 FFN at position $n + a(P(n)+1) + b - (p-h)$ and weight each of them by $2^{-h+(p-1)/2}$ and add them in one dimension. The Layer 4 FFN will calculate r when the first dimension of the input is 1 and $-r$ otherwise. Hence, for each $a \geq 0$, the $n + a\left(\mathrm{P}(n)+1\right) - hp, h \in [\Lambda : \Lambda + d]$ token contains a dimension i_1 which is the $k - \Lambda$ dimension of the output of the RNN at position a.

7. **Layer 5** Layer 5 Attention will have $d+1$ heads and each head will be of width $O(1)$. Head $h \in [d+1]$ will copy the dimension i_1 of the output of Layer 4 FFN at position $n + a(P(n)+1) + b - (h+\Lambda)p$ to a disjoint dimension i_{h+1}. The Layer 5 FFN will then make sure the output of Layer 5 satisfies the following:
   - When $b \neq \mathrm{P}(n)$, one dimension of the output will be n times the output of gate $s(b)$ when RNN circuit is calculating the output at $a + \mathbf{1}[b = \mathrm{P}(n)+1]$ position plus 1, which will decode to the corresponding value.
   - When $b = \mathrm{P}(n)$, the first d dimension of the output will be the same as the output of the RNN at position a, and the rest dimension will be 0, which will decode to the same token as the chain of thought of the RNN at position $a+1$.

This concludes the construction of the Transformer. □

B.4 Proof of Theorem 3.1

In this section, we will prove Theorem 3.1. We will first restate the theorem for convenience.

Theorem 3.1. *Given any constant A, B, for any polynomial-time Turing machine $T \in \mathrm{TIME}(n^A)$ with B states and vocabulary size B, there exists a retrieval augmented RNNs (see Definition A.9) with vocabulary of B special symbol, $O(A\log n + B)$ bit memory, and $O(B^2(A\log n + B))$ parameter, that can simulate the output of T on any input with length n with a chain of thought of length $O(n^A)$.*

Proof of Theorem 3.1. We will denote the state of T as $1, \ldots, B$ (we will use 1 as the initial state) and the vocabulary of T as $1, \ldots, B$. We will assume T operates on an infinite tape TAPE, which is a sequence of cells indexed by $\mathbb{Z}$. We will also assume that the tape is initialized with all cells being 0 except for the n cell starting at 1. The turing machine also has a pointer p that points to a cell in the tape. The pointer is initialized to 1. At each time step, the turing machine reads the cell pointed by $\mathrm{POINTER}$ and updates the cell pointed by $\mathrm{POINTER}$ and the pointer p according to the transition function $\delta : [B+1] \times [B] \to [B] \times [B] \times \{-1, 1\}$, which takes the current state and the current cell value (could be empty, which corresponds to $B+1$) as input and outputs the new state, the new cell value and the direction to move the pointer. The turing machine halts when the state is B. Because $T \in \mathrm{TIME}(n^A)$, the turing machine will always halt in n^A step. We will use $\mathrm{TAPE}[t, i]$ as the value on the i-th cell on TAPE before the t timestep. We will use $\mathrm{POINTER}[t]$ as the value of the pointer before the t timestep.

We will first define the sequence that the retrieval augmented RNN will generate and then construct an RNN that can generate such sequence.

Sequence generation. The input token sequence $\mathcal{S}_{\mathrm{in}}$ will be as followed,

$$\mathcal{S}_{\mathrm{in}} = \texttt{<s>}, 1, \mathrm{TAPE}[1,1], 1, 2, \mathrm{TAPE}[1,2], 2, \ldots, n, \mathrm{TAPE}[1,n], n$$

Here all the symbols on the tape are represented by one special symbol in the vocabulary. Given this input token sequence, the retrieval augmented RNN will generate the following output token sequence,

$$\begin{aligned}\mathcal{S} = \mathcal{S}_{\text{in}}, &\text{StartSearch}, (\text{POINTER}[1] \ \ \texttt{(.)} \quad \text{POINTER}[1]\texttt{.*\$}), \text{EndSearch},\\ &\text{SearchResult}(1),\\ &\text{POINTER}[1], \text{TAPE}[2, \text{POINTER}[1]], \text{POINTER}[1]\\ &\ldots\\ &\text{StartSearch}, (\text{POINTER}[t] \ \ \texttt{(.)} \quad \text{POINTER}[t]\texttt{.*\$}), \text{EndSearch},\\ &\text{SearchResult}(t)\\ &\text{POINTER}[t], \text{TAPE}[t+1, \text{POINTER}[t]], \text{POINTER}[t],\end{aligned}$$

Here $\text{SearchResult}(t)$ is defined as

$$\text{SearchResult}(t) = \begin{cases} \text{FAILED; if POINTER}[t] \text{ is empty cell before } t \\ \text{TAPE}[t, \text{POINTER}[t]]\text{;otherwise} \end{cases}$$

Clearly, the output token sequence simulates the turing machine T on the tape TAPE due to the following lemma.

Lemma B.16. *Given any $t \in [n^A]$ and $i \in [n^A]$, the last string in $\mathcal{S}$ that contains i, i as a substring is $i, \text{TAPE}[t, i], i$ if $\text{TAPE}[t, i]$ is not empty and is the empty string otherwise.*

Proof. The proof is by induction, for $t = 1$, the result clealy holds. For any $t \geq 2$, we only need to notice that $\text{POINTER}[t-1]$ is the only cell that can be updated at time $t-1$. □

Construction of RNN It is easy to see that the retrieval oracle will generate the correct $\text{SearchResult}(t)$ given the input $\mathcal{S}$/ Therefore, we will only need to construct an RNN that can generate the rest part of $\mathcal{S}$ given $\text{SearchResult}(t)$.

The construction of the RNN to generate $\mathcal{S}$ is as followed. We will first describe the state of RNN given any prefix $\mathcal{S}'$ of $\mathcal{S}$ and then describe the transition function necessary to generate the state.

The hidden state of RNN should contain the following dimension,

1. 1 dimension t' counting the number of StartSearch in $\mathcal{S}'$.
2. 1 dimension counting the number of EndSearch in $\mathcal{S}'$, disregarding the last token.
3. 1 dimension storing the state of the turing machine before t'.
4. 1 dimension storing the pointer of the turing machine before t'.
5. 1 dimension storing the state of the turing machine before $t' + 1$, when the first two dimensions are the same.
6. 1 dimension storing the pointer of the turing machine before $t' + 1$, when the first two dimensions are the same.
7. 1 dimension storing the value of $\text{TAPE}[t', \text{POINTER}[t']]$, when the first two dimensions are the same, and $t' \geq 1$.
8. $10(B + 10)$ dimensions storing the last 10 tokens In $\mathcal{S}'$.

Now we will describe the transition function necessary to generate the state. The last $10(B + 10)$ dimensions can be maintained trivially using a linear combination of the last $10(B + 10)$ dimension of the previous state and the embedding of the last token. The first 2 dimensions can be maintained by adding 1 to the first dimension if the last token is StartSearch and adding 1 to the second dimension if the third last token (which is maintained in the previous state) is EndSearch.

Now based on the last $10(B + 10)$ dimension, the RNN can determine which of the following cases the current state is in,

1. The current token is $\mathrm{StartSearch}$, in this case the model needs to update the third and fourth dimension to the fifth and sixth dimension of the previous state.
2. The current token is between $\mathrm{StartSearch}$ and $\mathrm{EndSearch}$ (including the $\mathrm{EndSearch}$), in this case the model don't need to update the first seventh dimension.
3. The current token is the search result. In this case, the model needs to update the fifth to seventh dimension. The seventh dimension is determined by the current token, which has an one-to-one correspondence with the cell value. The fifth and sixth dimension can be computed by a FFN with $O(B^2)$ width by taking state at time t', $\mathrm{POINTER}[t']$ and $\mathrm{TAPE}[t', \mathrm{POINTER}[t']]$ as input.
4. The current token is after search result and before the next $\mathrm{StartSearch}$. In this case, the state don't need to the model don't need to update the first seventh dimension.

Also, in all the position, the RNN has all the information needed to compute the next token and can implement the mapping in an $O(B^2)$-wide network. The proof is complete. ☐

B.5 Proof of Theorem 3.2

In this section, we will prove Theorem 3.2. We will first restate the theorem for convenience.

Theorem 3.2. *There exists a retrieval augmented RNNs with $O(\log n)$ bit memory and $O(\log n)$ parameter, that can solve IsTree of size n with a chain of thought of length $O(n)$.*

Proof of Theorem 3.2. We will first define the sequence that the retrieval augmented RNN will generate and then construct an RNN that can generate such sequence.

Sequence Generation. We will use a variant of Algorithm 1 to generate the sequence and we will use the concatenation of the tokenization of the sequence returned by Algorithm 2 as the sequence that the retrieval augmented RNN will generate.

RNN Construction. We can use similar proof in Theorem 3.1 by having the RNN to memorize local sequences and determine the phase of Algorithm 2 it is in. The RNN will maintain the length of S_2 and the top of S_1 in the state and it is easy to check that the retrieval function will retrieve the correct result for each search query. The way to determine the next vertex in the stack is the same as in the proof of Lemma B.11. We will omit the simple but tedious detailed construction here. ☐

Algorithm 2 Depth-First Search Algorithm with Retrieving

```
Require: A graph $G = (V, E)$ with $n$ vertices and $E$ has an ordering $e_1, \dots, e_m$.
 1: Initialize two stacks of vertices $S_1, S_2$ with $S_1 = [v_1], S_2 = \emptyset$, a list $L$ with $L = \emptyset$, and a vertex
    $v' =$ FAILED.
 2: while $S_1$ is not empty do
 3:   Let $v$ be the top of $S_1$. Push $v$ to $L$.
 4:   Generate the regular expression $r_2 =$ ((\d +) - v)
 5:   Let $f(v)$ be the predecessor of $v$ in $S_1$ for the first time and FAILED when $v = v_1$.
 6:   Push StartSearch, $r_2$, EndSearch, $f(v)$ to $L$.
 7:   if $v' \neq f(v)$ then
 8:     Generate the regular expression

        $r_1 =$ (v ~ v'|v' ~ v)(?:(?!v ~ \d +|\d + ~ v).)*?(v ~ (\d +)|(\d +) ~ v)

 9:   else
10:     Generate the regular expression $r_1 =$ (v ~ (\d +)|(\d +) ~ v)
11:   end if
12:   Push StartSearch, $r_1$, EndSearch to $L$.
13:   if there exists a neighbor $u$ of $v$ such that $(u, v)$ has larger order than $(v, v')$ when $v' \neq f(v)$
      or there exists a neighbor $u$ of $v$ such that $u \neq f(v)$ when $v' = f(v)$ then
14:     Choose $u$ such that edge $(u, v)$ has the smallest possible order and push $u$ to $L$. Let $v'' = u$.
15:   else
16:     Push FAILED to $L$. Let $v'' =$ FAILED.
17:   end if
18:   if $v'' = f(v) \neq$ FAILED then
19:     Generate the regular expression

        $r_3 =$ (v ~ v''|v'' ~ v)(?:(?!v ~ \d +|\d + ~ v).)*?(v ~ (\d +)|(\d +) ~ v)

20:     Push StartSearch, $r_3$, EndSearch to $L$.
21:     if there exists a neighbor $u$ of $v$ such that $(u, v)$ has larger order than $(v, v'')$ then
22:       Choose $u$ such that edge $(u, v)$ has the smallest possible order and push $u$ to $L$. Let
          $v'' = u$.
23:     else
24:       Push FAILED to $L$. Let $v'' =$ FAILED.
25:     end if
26:   end if
27:   if $v'' =$ FAILED then
28:     Pop $v$ from $S_1$. Push $v$ to $S_2$. Let $v' = v$.
29:   else
30:     Generate the regular expression $r_4 =$ ((\d +) - v'')
31:     Push StartSearch, $r_4$, EndSearch, 0 to $L$.
32:     if $v''$ is not in $S_1$ then
33:       Push FAILED, $v$, $-$, $v''$ to $L$.
34:       Push $v''$ to $S_1$. Let $v' = v$.
35:     else
36:       Let $f(v'')$ be the predecessor of $v$ in $S_1$ for the first time and FAILED when $v'' = v_1$.
37:       Push $f(v'')$, NO to $L$.
38:       return $L$.
39:     end if
40:   end if
41: end while
42: if $S_2$ has $n$ vertices then
43:   Push YES to $L$.
44:   return $L$.
45: else
46:   Push NO to $L$.
47:   return $L$.
48: end if
```