What Can Transformer Learn with Varying Depth? Case Studies on Sequence Learning Tasks

Xingwu Chen [1] **Difan Zou** [2]

Abstract

We study the capabilities of the transformer architecture with varying depth. Specifically, we designed a novel set of sequence learning tasks to systematically evaluate and comprehend how the depth of transformer affects its ability to perform memorization, reasoning, generalization, and contextual generalization. We show a transformer with only one attention layer can excel in memorization but falls short in other tasks. Then, we show that exhibiting reasoning and generalization ability requires the transformer to have at least two attention layers, while context generalization ability may necessitate three attention layers. Additionally, we identify a class of simple operations that a single attention layer can execute, and show that the complex tasks can be approached as the combinations of these simple operations and thus can be resolved by stacking multiple attention layers. This sheds light on studying more practical and complex tasks beyond our design. Numerical experiments corroborate our theoretical findings.

1. Introduction

Transformers (Vaswani et al., 2017) have been recognized as the most powerful model to achieve state-of-the-art performances in various deep learning tasks such as vision, natural language process, and decision making (Dosovitskiy et al., 2020; Brown et al., 2020a; Chen et al., 2021). Its superior performance makes it the most prevalent architecture for building universal foundation models (Touvron et al., 2023; Dosovitskiy et al., 2021; Devlin et al., 2018; Ying et al., 2021; Ouyang et al., 2022), its superiority in capability makes it one of the most widely used architectures for building universal foundation models (Bommasani et al., 2021; Brown et al., 2020a; Kaplan et al., 2020). A central module in the transformer is the attention layer, which performs nonlinear sequence-to-sequence mapping that allows each token to attend to several other tokens based on the semantic relationship. By stacking multiple attention layers, the transformer models have been observed to be surprisingly strong at performing memorization, understanding, and reasoning from the input sequences.

The remarkable empirical performance of transformer has triggered a series of theoretical studies, which aim to understand the working mechanism of transformer. For instance, some early attempts, including RASP (Weiss et al., 2021) and Tracr (Lindner et al., 2023), propose to interpret the transformer model by translating its mechanism into programming languages. However, their explanations are still hard to parse and difficult to help obtain quantitative characterizations on the transformer's capability. More recently, people has particularly focused on the capability of transformer in certain aspects, including its universal approximation power (Kajitsuka & Sato, 2023; Yun et al., 2020), data memorization capacity (Kajitsuka & Sato, 2023; Mahdavi et al., 2023), reasoning ability (Boix-Adsera et al., 2023; Fu et al., 2023), and in-context learning (ICL) (Xie et al., 2021; Garg et al., 2022; Bai et al., 2023; von Oswald et al., 2023; Wu et al., 2023).

However, these research primarily focuses on specific, simplified tasks that only utilize a subset of the transformer's capabilities. In practice, tasks often involve complex combinations of these simpler tasks, rendering them more challenging. Moreover, these studies often assume that the data is well-structured, aligning perfectly with the desired input-output token pairs. In practical scenarios, transformer inputs typically consist of general sequences, with tokens generated through human learning processes. Consequently, it remains unclear whether a given transformer model can effectively handle practical sequence-based tasks that require leveraging multiple aspects of the transformer's capabilities. Specifically, the capacity and limitations of the transformer architecture for addressing diverse sequence learning tasks remain uncertain.

In this paper, we aim to comprehensively understand the per-

[1]Department of Computer Science, The University of Hong Kong. [2]Department of Computer Science and Insititute of Data Science, The University of Hong Kong. Correspondence to: Difan Zou <dzou@cs.hku.hk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

formance of the attention-based transformer architecture by investigating whether and how certain tasks can be learned by transformers with varying depth (i.e. the number of attention layers). Specifically, we have designed four sequence learning tasks, including sequence classification, in-context question answering, template matching, and in-context template matching tasks, aiming at assessing and understanding the transformer's memorization, reasoning, generalization, and contextual generalization abilities. Notably, these tasks are correlated and purposely designed to incrementally become harder, based on which we can analyze how these abilities vary depending on the number of attention layers employed and characterize the mechanism of different attention layers. We have then conducted a systematic theoretical analysis to address two key research questions: (1) the minimum number of attention layers required for the transformer to perform the four tasks; and (2) the respective roles of the different attention layers in accomplishing these tasks. Our contributions to the field are summarized as follows:

- We propose a new set of sequence learning tasks specifically designed to assess the capabilities of transformers. In contrast to prior research that often concentrates on isolated tasks with well-structured input data, our tasks are systematic, interconnected, and more representative of real-world scenarios (the input data are general sequences generated from human's learning process). By leveraging these tasks, we can accurately evaluate the transformer's proficiency in key areas such as memorization, reasoning, generalization, and contextual generalization, and interpret the underlying mechanism of attention layers.
- We then theoretically assess the learning ability of transformer with varying numbers of attention layers by presenting both positive and negative results. In particular, we prove that the transformer with single attention layer can memorize but fails on other tasks. On the opposite, we show that two-layer transformer can successfully perform the reasoning and generalization tasks, and the transformer may need 3 layers to conduct contextual generalization. We further conduct numerical experiments to validate the theoretical results. These theoretical findings justify the need of more attention layers to accomplish more complicated tasks (that require multi-step reasoning and generalization), which aligns with the emergence phenomenon of transformer (Wei et al., 2022).
- We further provide some evidences regarding the working mechanism of transformer to accomplish the designed tasks. We show that the single attention layer can perform simple copying, parsing, matching, and mapping operations. Then stacking multiple attention layers can achieve the combinations of these operations, thus accomplish the harder tasks. In our experiments, we show that the attention maps of a trained transformer for different tasks are consistent with our findings. This could be of independent interest to understand how transformer tackle more complicated tasks in practice.

2. Related Work

Theoretical Understanding of Transformers Remarkable achievements of transformer leads to various theoretical attempts to understand its underlying mechanisms. These works approach the understanding of transformers from different angles. From a universal-approximation perspective, researchers have proven that transformers can approximate any sequence-to-sequence mapping under mild assumptions about the data distribution and target functions(Yun et al., 2020; Kajitsuka & Sato, 2023; Mahdavi et al., 2023; Takakura & Suzuki, 2023). In addition to mapping sequences, there is a line of work that investigates the transformer's ability to learn in context (von Oswald et al., 2023; Garg et al., 2023; Guo et al., 2023; Zhang et al., 2023), generalize on certain tasks (Boix-Adsera et al., 2023) and even perform complex instructions (Giannou et al., 2023; Liu et al., 2022). While these works provide useful perspectives on what transformers can do and propose possible mechanisms, they often involve more layers than what is typically used in practice or fall short in explaining real-world tasks involving discrete tokens and functions. Additionally, some works try to understand transformers from a computational perspective, offering valuable insights for understanding important properties such as chain of thought (Feng et al., 2024; Merrill & Sabharwal, 2023; Li et al., 2024). Although these works show the expressive power and limitations of well-structured transformers for certain tasks, the detailed analysis of expressive power in specific layers remains unclear.

Empirical Understanding of Transformers In addition to theoretical investigations, researchers have also attempted to understand the mechanisms of transformers through empirical analysis, such as interpreting trained transformers to derive human-readable representations (Lindner et al., 2023; Friedman et al., 2023; Weiss et al., 2021; Zhou et al., 2023), explaining transformers through probing techniques (Clark et al., 2019; Prabhu et al., 2022; Zou et al., 2023) or leveraging other large language models (Bills et al., 2023). However, due to the complexity of large language models, the explanations derived from these experiments are often complex and challenging to comprehend. Moreover, these empirical methods only provide insight into *how* the model accomplishes certain tasks, while the underlying mechanisms and the minimum requirements for transformers to learn such algorithms, such as the minimum number of layers and attention heads, remain elusive. In comparison to previous theoretical work, we introduce a practical setting that adapts discrete functions and data. Unlike using random features, we employ an approach that is more easily explainable. Furthermore, we aim to provide a theoretical explanation for why smaller models with fewer layers struggle with certain tasks, instead of relying solely on experimental results. To the best of our knowledge, this is the first study that compares and explains the limitations of small transformers.

3. Preliminaries

Notations. The set of indices from 0 to $n-1$ is denoted by $[n]$. Boldface upper-case $\mathbf{X}$ and lower-case $\mathbf{x}$ represent matrices and vectors, respectively. Specifically, we use $[\cdot]$ as Python index notation where $\mathbf{X}[i,:]$ refers to the i-th row of $\mathbf{X}$ and $\mathbf{X}[:,j]$ refers to the j-th column of $\mathbf{X}$. Similarly, $\mathbf{x}[i]$ refers to the i-th element of $\mathbf{x}$.

3.1. Attention-only Transformers

The transformer (Vaswani et al., 2017) is a neural network that can map a matrix $[\mathbf{x}_0,\ldots,\mathbf{x}_{n-1}]$ of size $d\times n$ to a sequence $[\mathbf{y}_0,\ldots,\mathbf{y}_{n-1}]$. In this work, we consider a transformer with L hidden layers and a classifier output layer:

$$\mathtt{TF} = \underbrace{f_{\mathtt{cls}}}_{\text{classifier}} \circ \underbrace{\mathtt{TF}_L \circ \cdots \circ \mathtt{TF}_1}_{L \text{ hidden layers}}. \tag{1}$$

Moreover, as mentioned previously, the objective of this work is to investigate the reasoning and generalization ability of the attention calculations in the transformer. Thus in each hidden layer, we choose to explode the MLP module as it performs token-wise operations that may introduce unnecessary distortion to our analysis (Zhang et al., 2017). Mathematically, given the representation matrix $\mathbf{H}^{(l)}\in\mathbb{R}^{d'\times n}$ in the $(l+1)$-th layer, where d' denotes the dimension of the hidden representations, the transformer layer $\mathtt{TF}_{l+1}$ with m attention heads computes:

$$\begin{aligned}\mathbf{H}^{(l+1)} &= \mathtt{TF}_{l+1}(\mathbf{H}^{(l)})\\ &= \mathbf{H}^{(l)} + \frac{1}{m}\sum_{i=1}^{m}\left[\mathbf{V}_i^{(l)}\sigma\big((\mathbf{Q}_i^{(l)})^\top\mathbf{K}_i^{(l)}\big)\right],\end{aligned} \tag{2}$$

where $\mathbf{Q}_i^{(l)}=\mathbf{W}_{Q_i}^{(l)}\mathbf{H}^{(l)}$, $\mathbf{K}_i^{(l)}=\mathbf{W}_{K_i}^{(l)}\mathbf{H}^{(l)}$, and $\mathbf{V}_i^{(l)}=\mathbf{W}_{V_i}^{(l)}\mathbf{H}^{(l)}$ are the query, key, and value computed by the i-th attention head with learned weight matrices $\mathbf{W}_{Q_i}^{(l)}$, $\mathbf{W}_{K_i}^{(l)}$, and $\mathbf{W}_{V_i}^{(l)}$ respectively. These weight matrices have dimensions $\mathbb{R}^{d'\times d'}$. Besides, $\sigma(z)$ is the activation function, which is set as the ReLU function $\sigma(z)=\max\{0,z\}$ in this work.

Classifier Layer: Given the representation generated by the last hidden layer $\mathtt{TF}_L$, i.e, $\mathbf{H}^{(L)}=(\mathbf{h}_0^{(L)},\ldots,\mathbf{h}_{n-1}^{(L)})$, we make use of its last column, i.e., $\mathbf{h}_{n-1}^{(L)}$ to obtain the final prediction $\mathbf{o}=\mathbf{W}_O\mathbf{h}_{n-1}^{(L)}\in\mathbb{R}^C$, where $\mathbf{W}_O\in\mathbb{R}^{C\times d'}$ is the weight matrix of the classifier layer. Notably, C represents the total number of labels, which can be seen as (1) the vocabulary size for the token prediction task; or (2) the number of classes for the sequence classification task. The prediction result is then achieved by finding the index of the maximum entry of $\mathbf{o}$, i.e., $\hat{y}=\arg\max_{i\in[C]}\mathbf{o}[i]$.

Positional Encoding and Padding: Given a sequence of discrete tokens, denoted by $\mathbf{X}=[\mathbf{x}_0,\ldots,\mathbf{x}_{n-1}]\in\mathbb{R}^{d\times n}$, the initial representation of each token is composed by the original token embedding, positional encoding, and padding. In particular, note that the hidden dimension is d', the initial representation matrix for the sequence $\mathbf{H}^{(0)}$ is given by:

$$\mathbf{H}^{(0)} = \begin{pmatrix}\mathbf{x}_0 & \cdots & \mathbf{x}_{n-1}\\ \mathbf{p}_0 & \cdots & \mathbf{p}_{n-1}\\ \mathbf{0} & \cdots & \mathbf{0}\end{pmatrix}\begin{matrix}\to d\times n\\ \to n\times n\\ \to (d'-n-d)\times n\end{matrix}. \tag{3}$$

For the simplicity of analysis, we consider the one-hot positional encoding, i.e., we set $\mathbf{p}_i=[\mathbf{0}_i,1,\mathbf{0}_{n-(i+1)}]^\top$ for position i.

4. Memorization, Reasoning, and Generalization Tasks for Sequences

In this section, we will introduce the tasks designed to assess and understand the capability of transformers for tackling sequences. In particular, four tasks will be designed, which aim to characterize the capability of transformer structure in terms of memorization, reasoning, generalization, and contextual generalization.

4.1. Memorization: Sequence Classification Task

The memorization capability serves as a fundamental theoretical property for transformers. We start our understanding of the transformer model by characterizing its memorization capability. In particular, we consider the sequence classification task, as shown in Figure 1, one of the most important and successful tasks for transformer-based models (Devlin et al., 2018). To formulate the sequence classification task, we define the dataset $\mathcal{D}_{\mathtt{SC}}$ as a collection of N sequence-label examples, each with a different class type. Specifically, $\mathcal{D}_{\mathtt{SC}}=\{(\mathbf{X}^{(0)},y^{(0)}),\ldots,(\mathbf{X}^{(N-1)},y^{(N-1)})\}$, where $\mathbf{X}\in\mathbb{R}^{d\times n}$ is a sequence consisting of n discrete tokens from a word alphabet $\mathcal{X}$, and the corresponding labels $y^{(0)},y^{(1)},\ldots,y^{(N-1)}$ are distinct integer. Before input into the model, we first append a $\mathtt{CLS}$ token $\mathbf{c}$ at the end of each sequence, which is widely applied in transformer-based models as a representation of the whole sequence. Then this task is to characterize whether the transformer model can successfully map each sequence to the corresponding label based on the representation corresponding to the last token of the sequence, i.e., $\mathtt{CLS}$.

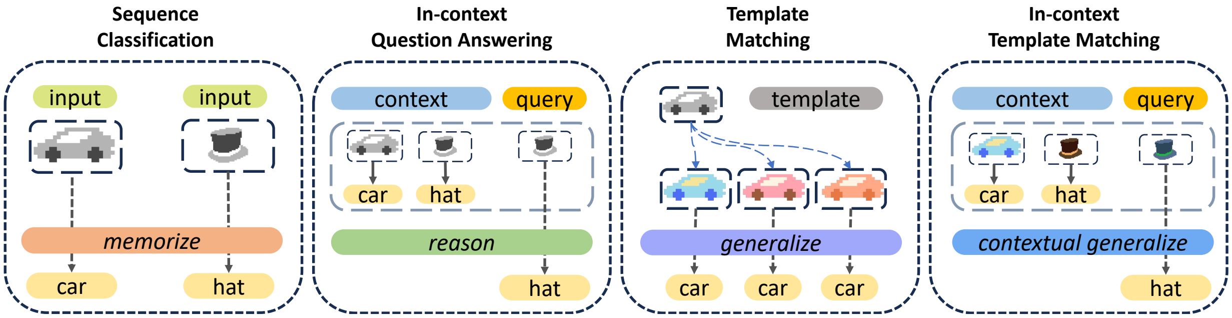

Figure 1: Descriptions of the four sequence learning tasks considered in this work, including (1) sequence classification task; (2) in-context question answering task; (3) template matching task; and (4) in-context template matching task. Here each input, context, and query are represented as sequences consisting of multiple tokens.

4.2. Reasoning: In-context Question Answering Task

In-context learning (Brown et al., 2020b) refers to the capability of model to learn from the context and provide answers to questions based on examples and their corresponding solutions. To characterize the reasoning capability of the transformer, we consider a simplified in-context learning task, called in-context question answering task, which is summarized in Figure 1. We consider a simple in-context learning problem with several question-answer pairs, the model is required to retrieve the corresponding answer based on the given question from the context.

To formulate our in-context question-answering task, we define three types of tokens: *question tokens* $\mathcal{Q} = \{\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_{n_q-1}\}$, *response sign* $\mathcal{R} = \{\mathbf{r}\}$, and *answer tokens* $\mathcal{A} = \{\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{n_a-1}\}$. Additionally, we use $\pi_0, \ldots, \pi_{n-1}$ and $\pi'_0, \ldots, \pi'_{n-1}$ denote the indices of the sampled question and answer tokens, π and π' correspond to permutations. The *response sign* is a special tag widely used in large language models like Llama2 (Touvron et al., 2023) and Galactica (Taylor et al., 2022) for guiding the model's behavior in question-answer scenarios. Our data is constructed as follows: we sample k questions from $\mathcal{Q}$, denoted as $(\mathbf{q}_{\pi_0}, \ldots, \mathbf{q}_{\pi_{k-1}})$ and k answers $(\mathbf{a}_{\pi'_0}, \ldots, \mathbf{a}_{\pi'_{k-1}})$ from $\mathcal{A}$. We then add the response sign $\mathbf{r}$ between each $\mathbf{q}_{\pi_i}$ and $\mathbf{a}_{\pi'_i}$, resulting in a context block length of $3k$: $\mathbf{B}^{(\pi,\pi')}_{\texttt{IC-QA}} = (\mathbf{q}_{\pi_0}, \mathbf{r}, \mathbf{a}_{\pi'_0}, \ldots, \mathbf{q}_{\pi_{k-1}}, \mathbf{r}, \mathbf{a}_{\pi'_{k-1}})$. Next, we randomly choose the question $\mathbf{q}_{\pi_c}$ (where $c \in [k]$) from the context block, and concatenate it with the final response sign: $[\mathbf{B}^{(\pi,\pi')}_{\texttt{IC-QA}}; \mathbf{q}_{\pi_c}; \mathbf{r}]$. We denote this data as $\mathbf{E}^{(\pi,\pi')}_{\texttt{IC-QA}}(c)$, a sequence length $n = 3k + 2$. Instead of pre-defining question-answer pairs, we consider each question to have n_a possible answers by choosing different permutation π and π', as the objective is to investigate whether the model can learn to retrieve answers from the context, rather than memorizing the question-answer pairs. In this way, with n_q questions and n_a answers, we can construct a dataset (denoted as $\mathcal{D}^{(k)}_{\texttt{IC-QA}}$) with $A^k_{n_q} \cdot A^k_{n_a} \cdot k$ examples, where $A^m_n = \frac{n!}{(n-m)!}$ denotes the number of ways to choose m elements from a set of n elements. Then, the task is to characterize that given any context block $\mathbf{E}^{(\pi,\pi')}_{\texttt{IC-QA}}(c) = [\mathbf{B}^{(\pi,\pi')}_{\texttt{IC-QA}}; \mathbf{q}_{\pi_c}; \mathbf{r}]$, whether the transformer can correctly output the desired answer $\mathbf{a}_{\pi'_c}$.

4.3. Generalization: Template Matching Task

Motivated by the learning process of humans, where we abstract new things into different patterns for further analysis. Inspired by the template task designed in Boix-Adsera et al. (2023) for studying the generalization ability of transformer, we consider a similar template matching task to investigate whether transformers with varying attention layers have the ability to generalize. In particular, we first deliver the formal definition of the template.

Definition 4.1. A **template** is a string $\boldsymbol{t} \in \mathcal{W}^l$, where $\mathcal{W}$ is an alphabet of "wildcards". A **substitution map** is an injection function $s : \mathcal{W} \to \mathcal{X}$ that maps wildcards to real word symbols $\mathcal{X}$. Here, $\mathcal{X}$ can be seen as the alphabet of tokens in language or pixel blocks in an image. Different wildcards should be mapped to different tokens to ensure that each sequence can be mapped to one and only one template. We write $\texttt{sub}(\boldsymbol{t}, s) \in \mathcal{X}^l$ for the sequence where each wildcard is substituted with the corresponding token: $\texttt{sub}(\boldsymbol{t}, s)_i = s(t_i)$. A **template labeling mapping** is a mapping from a template to the class index $f : \mathcal{W}^n \to \mathbb{Z}^*$.

In general, the template can be understood as the abstract concept of the data, i.e., in Figure 1, "car" is the concept of the car image with different colors. To construct our dataset, we first define a template set with all possible templates of length $n_{\texttt{tmpl}}$: $\mathcal{T} = \{\boldsymbol{t}_0, \boldsymbol{t}_1, \ldots, \boldsymbol{t}_{n_{\texttt{tmpl}}-1}\}$. We then use a *template labeling mapping* f to map each template $\boldsymbol{t}_i$ to a class y_i. After that, we use a set of $n_{\texttt{map}}$ *substitution maps* $\mathcal{S} = \{s_0, s_1, \ldots, s_{n_{\texttt{map}}-1}\}$ to generate data from the template to a real word sequence. We write the dataset as $\mathcal{D}_{\texttt{tm}} = \{(\texttt{sub}(\boldsymbol{t}_i, s_j), y_i)) : i \in [n_{\texttt{tmpl}}], j \in [n_{\texttt{map}}]\}$, where $y_i = f(\boldsymbol{t}_i)$ denotes the template label and $\texttt{sub}(\boldsymbol{t}, s)$ denotes the sequence of real-word symbols that follow the template $\boldsymbol{t}$ and token mapping function s.

Similar to the sequence classification task, we also append a CLS token at the end of the input sequence for generating the prediction. However, to investigate the generalization ability, the transformer cannot simply memorize all possible sequences but requires to learn their abstract patterns, i.e., the templates, to make the correct prediction. Then, the task is to characterize that given a sequence generated via $\mathrm{sub}(\boldsymbol{t}_k, s_i)$, whether the transformer can identify the template $\boldsymbol{t}_k$ and output the correct prediction y_k. We call the model can *generalize on template* $\boldsymbol{t}_k$ if it can correctly predict all possible sequences generated by $\boldsymbol{t}_k$ and s_i.

4.4. Contexture Generalization: In-context Template Matching Task

We then consider a more complex and general problem that is designed as the combination of in-context question-answering and template matching tasks, which requires the model to perform both reasoning and generalization simultaneously. This task is summarized in Figure 1.

In particular, we formulate our problem by replacing the question in the context block $\mathbf{B}_{\texttt{IC-QA}}$ from a simple token $\mathbf{q}$ to the template data $\mathrm{sub}(\boldsymbol{t}, s)$. To construct our dataset, we need to define a set of templates $\mathcal{T} = \{\boldsymbol{t}_0, \boldsymbol{t}_1, \ldots, \boldsymbol{t}_{l_{\texttt{tmpl}}-1}\}$. All templates have the same length l. Rather than predefining a mapping from the template $\boldsymbol{t}$ to a class label y, we follow the construction process in the previous in-context question-answering task. We first randomly choose k templates from $\mathcal{T}$ and k answers from the answer token set $\mathcal{A}$: $(\boldsymbol{t}_{\pi_0}, \ldots, \boldsymbol{t}_{\pi_{k-1}})$ and $(\mathbf{a}_{\pi'_0}, \ldots, \mathbf{a}_{\pi'_{k-1}})$. Then we can consider k different substitution mapping function $s_{\pi''_0}, \ldots, s_{\pi''_{k-1}}$ for $\boldsymbol{t}_{\pi_0}, \ldots, \boldsymbol{t}_{\pi_{k-1}}$ to generate sequences of real-world symbols, denoted as $\mathbf{X}_{\pi_0}, \ldots, \mathbf{X}_{\pi_{k-1}}$, where $\mathbf{X}_{\pi_i} = \mathrm{sub}(\boldsymbol{t}_{\pi_i}, s_{\pi''_i})$. Then, the context block is defined as $\mathbf{B}^{(\pi,\pi')}_{\texttt{IC-TM}} = \big(\mathbf{X}_0, \mathbf{r}, \mathbf{a}_{\pi'_0}, \ldots, \mathbf{X}_{\pi_{k-1}}, \mathbf{r}, \mathbf{a}_{\pi'_{k-1}}\big)$. Then we randomly choose a query template $\boldsymbol{t}_{\pi_c}$ with $c \in \{0, \ldots, k-1\}$ and use a new mapping function $s_{\pi''_k}$ to get the sequence of real-world symbols $\mathbf{X}_{\pi_c}$. Then, the entire input sequence is defined as $\mathbf{E}^{(\pi,\pi')}_{\texttt{IC-TM}}(c) = [\mathbf{B}^{(\pi,\pi')}_{\texttt{IC-TM}}, \mathbf{X}_{\pi_c}, \mathbf{r}]$, and the desired answer should be $\mathbf{a}_{\pi'_c}$. Then, the entire dataset, denoted as $\mathcal{D}_{\texttt{IC-TM}}$, is the collection of all sequences-answer pairs that generated by using all possible templates, answers, and mapping functions.

Compared with the in-context learning question-answering and template matching tasks, this task requires the model to reason from the context and generalize to the unseen data. For instance, in Figure 1, the model needs to first identify the template/concept of the query image (which is "hat"), and then seeks the answer from the context (there is an example image using the same template and providing the answer "hat"). In this task, the model should capture the similarity between each question (generalization) and retrieve the answer from the context (reasoning).

Summary and Discussion. We provide data examples and a more detailed comparison for the four tasks in Appendix A. Note that we employ these tasks to assess the model's capacity, i.e. for the given architecture, especially the transformer with different attention layers, *what* the model can do and *how* the model do it. Specifically, we aim to determine whether there exists a particular configuration of the transformer model, such that all examples in the dataset can be perfectly learned. This ability is independent of the training process; our focus is solely on the ability of the transformer's architecture for tackling these tasks.

5. Main Results

In this section, we present our main findings regarding the aforementioned tasks. We will focus on characterizing how transformer model performs on these tasks with varying attention layers. We will prove both negative and positive results on the capability of transformer when different numbers of attention layers are stacked.

5.1. Single-Layer Transformer Can Memorize

We commence our investigation by examining the memorization capability of a single-layer transformer. In this scenario, the model's objective is to accurately classify N sequences with distinct labels. In particular, we will show that given sufficient heads, a single-layer transformer has the capability to memorize all data points. We summarize this result in the following Theorem.

Theorem 5.1. *For any dataset of the sequence classification task, denoted by D_{SC}, let d be the token dimension, and n be the length of the sequence (i.e., number of tokens). Then there exists a transformer TF with $L = 1$ attention layer, n attention heads, and model embedding dimension $d' = \max\{nd, d+n\}$ such that for all $(\mathbf{X}, y) \in \mathcal{D}_{SC}$, it holds that $TF(\mathbf{X}) = y$*[1].

We first remark that the goal of Theorem 5.1 is to demonstrate the ability of the single-layer transformer for the memorization task, while the (horizontal) model size, i.e., number of heads and embedding dimensions, are not optimized. It is possible to further sharpen our analysis, e.g., applying the techniques in Mahdavi et al. (2023), to relax the conditions on the (horizontal) size of the transformer model.

To achieve this, we show that a single attention layer, with n attention heads, can perform the **mapping** operation to transformer the input sequence, formulated as a matrix of embeddings (of dimension $\mathbb{R}^{d\times n}$), to a distinct vector representation. Moreover, we show that these vector representations are linearly independent. Then the output classifier

[1]Here we slightly abuse the notation use $\texttt{TF}(\mathbf{X})$ the denote the prediction result of the input $\mathbf{X}$. Similar notations will be used in other theorems.

layer, equipped with the weight matrix $\mathbf{W}_O \in \mathbb{R}^{N \times d'}$ (N denotes the number of total labels), can map each vector representation to a probability vector, where the index of the largest entry corresponds to the desired sequence label. The full proof and construction of the transformer weights can be found in Appendix E.

Theorem 5.1 demonstrates that the one attention layer is sufficient for memorization. However, it is important to note that memorization alone cannot guarantee other more challenging and critical abilities such as reasoning and generalization. Characterizing the ability of transformer in these aspects will be the focus of the subsequent subsections.

5.2. Two-Layer Transformer Performs Reasoning

Then we explored the ability of transformers to reason using simple in-context learning tasks. Previous research has investigated similar tasks, using induction heads (Olsson et al., 2022) and transformer circuits (Elhage et al., 2021), to assess the transformer's reasoning ability. However, the theoretical basis for these observations remains unclear, the connection between the number of attention layers and the reasoning ability has not been thoroughly studied.

In this section, we theoretically characterize the reasoning performance of single-layer and two-layer transformer models on the in-context question-answering task. First, we provide the following theorem to show that any single-layer transformer cannot perfectly perform the reasoning task.

Theorem 5.2. *Let $\mathcal{D}_{\mathtt{IC-QA}}$ be a dataset of the in-context question-answering task and n be the number of question-answer pairs. Then for any transformer with $L = 1$ attention layer, no matter how many heads are applied, there exists at least one data point $(\mathbf{E}^{(\pi,\pi')}_{\mathtt{IC-QA}}(c), \mathbf{a}_{\pi'_c}) \in \mathcal{D}^{(k)}_{\mathtt{IC-QA}}$ such that $\mathtt{TF}\big(\mathbf{E}^{(\pi,\pi')}_{\mathtt{IC-QA}}(c)\big) \neq \mathbf{a}_{\pi'_c}$.*

Theorem 5.2 suggests for any single-layer transformer, there must exist at least one data point that cannot be correctly predicted, suggesting its inability to perfectly tackle the in-context question-answer task. The idea to prove this is to show that single-layer attention function can preserve the linear dependency (defined in terms of the set operations, see Appendix D). In other words, if multiple input sequences, such as the entire dataset, exhibit some dependence, then the corresponding outputs of the single-layer attention will also display linear dependence. By leveraging this linear dependence in the outputs, we can demonstrate that the attention function fails to successfully learn all question-answering tasks. The detailed proof can be found in Appendix F.1.

Moreover, we claim that Theorem 5.2 is not limited to the ReLU attention, but can also apply to softmax attention when using single head inAppendix I (extending to multiple head case is left for future study). Then, we show that, in the following theorem, a two-layer transformer can resolve the issue of the first-layer transformer and perfectly reason all sequences in the dataset.

Theorem 5.3. *For any dataset of the in-context question-answering task, denoted by $\mathcal{D}_{\mathtt{IC-QA}}$, let k be the number of question-answer pairs (the sequence length is $n = 3k + 2$) and d be the dimension of the token embedding. There exists a transformer $\mathtt{TF}$ with $L = 2$ attention layers, 1 attention head, and $d' = d + n$ such that for all $\big(\mathbf{E}^{(\pi,\pi')}_{\mathtt{IC-QA}}(c), \mathbf{a}_{k'_c}\big) \in \mathcal{D}_{\mathtt{IC-QA}}$, it holds that $\mathtt{TF}(\mathbf{E}^{(\pi,\pi')}_{\mathtt{IC-QA}}(c)) = \mathbf{a}_{k'_c}$.*

Our proof, i.e., the construction of such a two-layer transformer model, draws inspiration from (Friedman et al., 2023), which shows that the two-layer transformer can perform a **copying-matching** procedure to accomplish the template matching task. We construct the first layer to perform the **copying** operation among question and answer tokens to aggregate each question and the corresponding answer together. The second layer is implemented as an induction head (Olsson et al., 2022) to perform the **matching** operation between the token representations (which already aggregate the question and answer together) with the same question (i.e., the query question), and then output the desired answer. The detailed construction is in Appendix F.2.

By combining Theorem 5.2 and Theorem 5.3, we can conclude that it requires two attention layers to perfectly perform the reasoning. However, the 2-layer attention-only transformer can do more than just copying and matching. Next, we will show that 2-layer transformers can also accomplish the generalization task through a different mechanism.

5.3. Two-Layer Transformer Can Generalize

In this part, we shift our focus to the generalization ability of transformers. Specifically, we consider the template matching task, where each template has a distinct label, and sequences that follow from the same template will be assigned by the same label. Our goal is to investigate whether and how transformers can successfully perform this task, i.e., identify the template of the input sequence and predict its label, for all possible sequences. This serves as the necessary condition for the generalization of transformer (Boix-Adsera et al., 2023). Similar to the findings in Section 5.2, we also observe that a single-layer transformer fails to accurately learn this generalization process, which is summarized in the following theorem.

Theorem 5.4. *Let $\mathcal{D}_{\mathtt{TM}}$ be a dataset of the template matching task and n be the sequence length. Then for any transformer with $L = 1$ attention layer, no matter how many heads are applied, there exists at least one data $(\mathtt{sub}(\boldsymbol{t}, s), y) \in \mathcal{D}_{\mathtt{TM}}$, generated via a template $\boldsymbol{t}$ and a mapping s, such that $\mathtt{TF}(\mathtt{sub}(\boldsymbol{t}, s)) \neq y$.*

We follow a similar idea for proving 5.2 to prove the above argument. In particular, we can show that there are two templates such that all possible sequences generated accordingly are linearly dependent. Then if these two templates have different labels, the single-layer transformer fails to correctly classify all sequences generated via these two templates. The detailed proof can be found in Appendix G.1.

This intriguing result suggests that although single-layer transformers possess strong memorization abilities, they struggle with more complex tasks. Moreover, we show that this template matching task can be performed by a two-layer transformer, which is stated in the following theorem.

Theorem 5.5. *For any dataset of the template matching task, denoted by $\mathcal{D}_{TM}$, let n be the sequence/template length and d be the token embedding dimension. Then there exists a transformer $\texttt{TF}$ with $L = 2$ attention layers, 1 attention heads, and $d' = d+n$ such that for all $(\texttt{sub}(\boldsymbol{t}, s), y) \in \mathcal{D}_{TM}$, it holds that $\texttt{TF}(\texttt{sub}(\boldsymbol{t}, s)) = y$.*

We show that such a two-layer transformer can be constructed using a **parsing-mapping** process. In particular, the first layer can be designed to parse the sequence into the corresponding template, then the second layer can perform a memorization process that is similar to the sequence classification task investigated in Section 5.1. These findings prompt us to reconsider the mechanism of multi-layer transformers, instead of solely relying on memorizing all the data (Yun et al., 2020). The detailed construction can be found in Appendix G.2.

5.4. Three-Layer Transformer Can Perform Contextual Generalization

In previous sections, we have shown that a 2-layer transformer is capable of conducting reasoning and generalization tasks. Now we will focus on a more challenging in-context template matching task that requires the model to perform generalization and reasoning simultaneously, i.e., exhibiting the contextual generalization capability.

First, since the in-context template matching task can degenerate to the standard in-context question-answering task (e.g., using identity mapping from the template alphabet to real-world symbols). Then, we can straightforwardly leverage the result in Theorem 5.2 to demonstrate the failure of the single-layer transformer in accomplishing this task. Moreover, note that when tackling the in-context question-answering and template matching tasks, the transformer is constructed to perform two-step copy-matching and parse-mapping procedures, respectively. Therefore, regarding the in-context template matching task, we can design a transformer to perform a three-step **parsing-copying-matching** procedure, which is constructed using three attention layers. We state this result in the following theorem.

Theorem 5.6. *For any dataset of the in-context template matching task, denoted by $\mathcal{D}_{IC\text{-}TM}$, let l be the template length, k be the number of question-answer pairs (then the sequence length is $n = k(l+2)+l+1$), and d be the dimension of the token embedding. There exists a transformer $\texttt{TF}$ with $L = 3$ attention layers, $2l$ attention heads, and $d' = d + n + l + 2$ such that for all $\left(\mathbf{E}^{(\pi,\pi')}_{IC\text{-}TM}(c), \mathbf{a}_{k'_c}\right) \in \mathcal{D}_{IC\text{-}TM}$, it holds that $\texttt{TF}(\mathbf{E}^{(\pi,\pi')}_{IC\text{-}TM}(c)) = \mathbf{a}_{k'_c}$.*

We remark that Theorem 5.6 does not imply that the in-context template matching task cannot be accomplished by two-layer transformers. However, in our numerical experiments (see Figure 2), we find that the two-layer transformer struggles with this task and can even not perform well during the training. Therefore, we tend to believe that a three-layer transformer may be the shallowest one to perform contextual generalization, while the rigorous proof for the failure of two-layer models is left for future study.

6. Experiments

In this section, we verify the main results presented in Section 5 through synthetic datasets. We examine the accuracy and loss dynamics for the four tasks across different layers and heads of transformers. Additionally, we study the reasoning and generalization mechanisms of transformers by analyzing attention maps and comparing them with our constructed transformer. The detailed experimental setup is presented in Appendix B.

6.1. The Impact of Attention Layers on Different Tasks

We begin by studying the impact of transformer depth on these tasks. The results are shown in Figure 2. We observe that a single-layer transformer performs well on the memorization task but struggles with tasks related to generalization and reasoning. This validates Theorems 5.1, 5.2, 5.3, 5.4, and 5.5. Single-layer transformer performs like a random guess for generalization and reasoning tasks.

For the contextual generalization task, we interestingly find the same random guessing degeneration in both single-layer and two-layer transformers, indicating that a 2-layer transformer might not be able to handle such a complex task that requires both generalization and reasoning. Instead, a 3 layer transformer performs perfectly on this task. This validates Theorem 5.6. Through this task, we can observe the emergence of more complex reasoning and generalization when we extend the layer of the transformer from 2 to 3. In Appendix B, we also study the performance of a 4-layer transformer, which shows that compared to a 3-layer transformer, a 4-layer transformer can perform contextual generalization more quickly. This emphasizes the effectiveness of using deeper model to perform harder tasks.

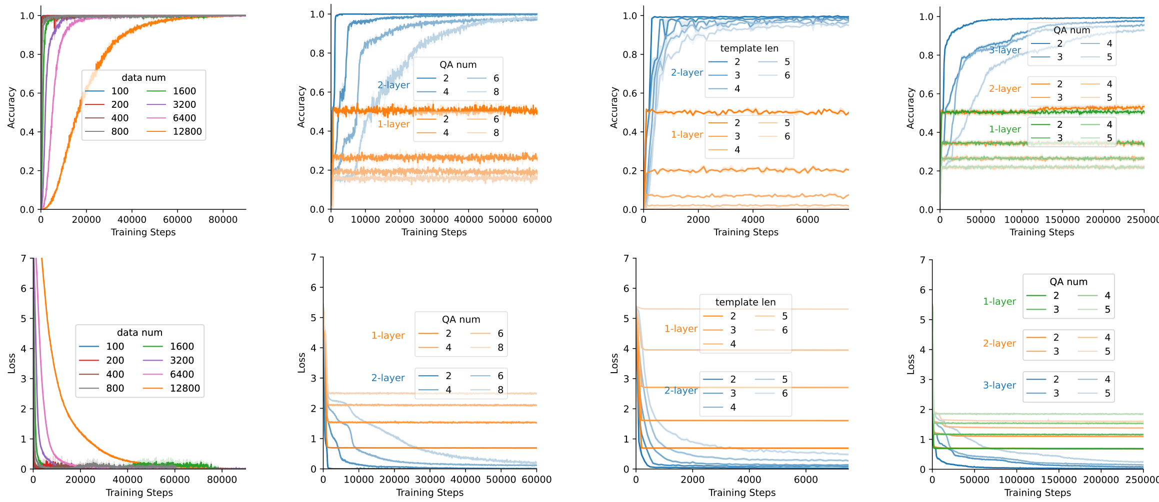

Figure 2: Performance of different layers of transformers on memorization, reasoning, generalization, and contextual generalization tasks. *Far left column*: A single-layer transformer can memorize sequences with distinct labels. *Center left column*: A single-layer transformer struggles with reasoning tasks, while a two-layer transformer can learn reasoning with enough training steps. *Center right column*: A single-layer transformer struggles with generalizing on template tasks, while a two-layer transformer can quickly grasp the method for generalization. *Far right column*: When it comes to more complex contextual generalization tasks, a 1/2-layer transformer fails, but a 3-layer transformer can perform well on such tasks.

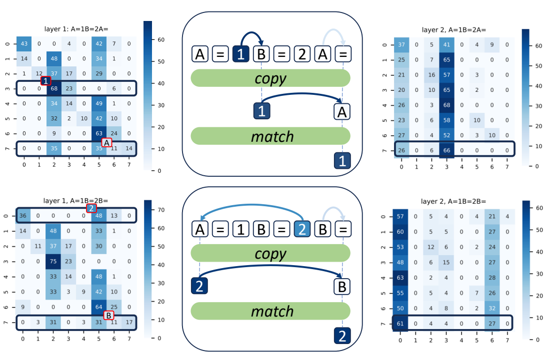

Figure 3: Attention maps for a trained two-layer transformer in the reasoning sequences "A=1B=2A=" (*top row*) and "A=1B=2A=" (*bottom row*).

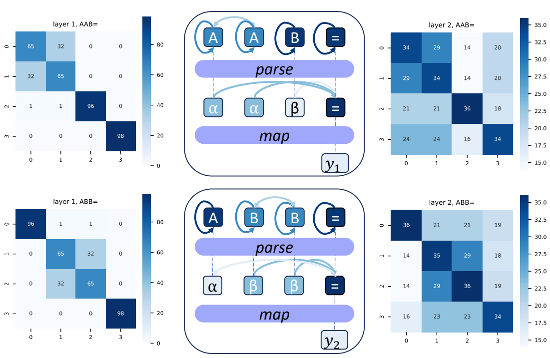

Figure 4: Attention maps for a trained two-layer transformer in the template sequences "AAB=" (*top row*) and "ABB=" (*bottom row*).

6.2. Algorithms Behind Trained Transformers

To further understand how transformers achieve generalization and reasoning, we analyze the attention maps for some typical examples. The results show that trained transformers exhibit similar mechanisms to our constructions.

In the reasoning task, we observe operations in Figure 3 that are similar to the constructed copying-matching mechanism. In particular, we can identify two "copying" operations (the corresponding value in the attention map is relatively high) in the first layer: in Figure 3 (*top row*), tokens "1" and "A" are copied to the 4-th and last positions respectively. In the second layer, we can then identify a "matching" operation: token "1", which now appears in the 3-th position, strongly correlates with the token "A", which now appears in the last position. This further leads to the correct answer "1". Similar observation can be found in the second example: token "2" and "B" are copied to the first and last positions respectively in the first layer; then a matching between the token "B" in the last position and the value "2" in the first position occurs, leading to the correct answer.

In the template matching task, we also find evidence of our constructed parsing-mapping mechanism in Figure 4. Specifically, a "parsing" operation that checks the similarity of tokens in other positions can be observed in the first layer: for input sequence "AAB=", the repeat token "A" in positions 0 and 1 share attention with each other, for the input "ABB=", the repeat token in positions 1 and 2 share the position information. In this way, the model parses the input sequence into a template representation, $\texttt{AAB} \to \alpha\alpha\beta$ and $\texttt{ABB} \to \alpha\beta\beta$, which can be mapped to different templates by utilizing the memorization ability of the transformer.

7. Discussion

In this study, we explore the capabilities of transformers with varying attention layers in performing various tasks, including memorization, reasoning, generalization, and contextual generalization. Our investigation reveals the limitations of single-layer transformers when dealing with complex tasks, and highlights the importance of using multiple attention layers to achieve optimal performance in reasoning, generalization, and contextual generalization tasks. Our findings shed light on the theoretical properties of transformer models, offering insights into their design and optimization for diverse tasks. Besides, our framework can be further expanded to more challenging tasks. For example, we can expand our in-context QA as "nested in-context QA" task, where the model must perform a chain-of-thought process to arrive at the final answer. This could involve a sequence like "a → b b → c c → d a → d", where we can design a transformer with 6 layers that performs "copy-matching" 3 times to solve this problem effectively. We believe that expanding our four tasks is worth for further investigation, and our framework can offer valuable insights. Moreover, our analysis, which determines the threshold for attention layers in solving complex tasks, can even offer explanations for the scalability of transformers (Kaplan et al., 2020) and their emergent abilities (Wei et al., 2022).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgement

We would like to thank the anonymous reviewers and area chairs for their helpful comments. XC and DZ are supported by NSFC 62306252 and the central fund from HKU IDS.

References

Bai, Y., Chen, F., Wang, H., Xiong, C., and Mei, S. Transformers as Statisticians: Provable In-Context Learning with In-Context Algorithm Selection, July 2023.

Bills, S., Cammarata, N., Mossing, D., Tillman, H., Gao, L., Goh, G., Sutskever, I., Leike, J., Wu, J., and Saunders, W. Language models can explain neurons in language models. *URL https://openaipublic. blob. core. windows. net/neuron-explainer/paper/index. html.(Date accessed: 14.05. 2023)*, 2023.

Boix-Adsera, E., Saremi, O., Abbe, E., Bengio, S., Littwin, E., and Susskind, J. When can transformers reason with abstract symbols?, October 2023.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020a.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-Shot Learners, July 2020b.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Clark, K., Khandelwal, U., Levy, O., and Manning, C. D. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Feng, G., Zhang, B., Gu, Y., Ye, H., He, D., and Wang, L. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36, 2024.

Friedman, D., Wettig, A., and Chen, D. Learning Transformer Programs. *arXiv preprint*, 2023.

Fu, H., Guo, T., Bai, Y., and Mei, S. What can a single attention layer learn? a study through the random features lens. *arXiv preprint arXiv:2307.11353*, 2023.

Garg, S., Tsipras, D., Liang, P. S., and Valiant, G. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

Garg, S., Tsipras, D., Liang, P., and Valiant, G. What Can Transformers Learn In-Context? A Case Study of Simple Function Classes, August 2023.

Giannou, A., Rajput, S., Sohn, J.-Y., Lee, K., Lee, J. D., and Papailiopoulos, D. Looped transformers as programmable computers. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/giannou23a.html.

Guo, T., Hu, W., Mei, S., Wang, H., Xiong, C., Savarese, S., and Bai, Y. How Do Transformers Learn In-Context Beyond Simple Functions? A Case Study on Learning with Representations, October 2023.

Kajitsuka, T. and Sato, I. Are Transformers with One Layer Self-Attention Using Low-Rank Weight Matrices Universal Approximators?, July 2023.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling Laws for Neural Language Models, January 2020.

Li, Z., Liu, H., Zhou, D., and Ma, T. Chain of thought empowers transformers to solve inherently serial problems. *arXiv preprint arXiv:2402.12875*, 2024.

Lindner, D., Kramár, J., Rahtz, M., McGrath, T., and Mikulik, V. Tracr: Compiled transformers as a laboratory for interpretability. *arXiv preprint arXiv:2301.05062*, 2023.

Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., and Zhang, C. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2022.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

Mahdavi, S., Liao, R., and Thrampoulidis, C. Memorization Capacity of Multi-Head Attention in Transformers, October 2023.

Merrill, W. and Sabharwal, A. The expresssive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Prabhu, V., Yenamandra, S., Singh, A., and Hoffman, J. Adapting self-supervised vision transformers by probing attention-conditioned masking consistency. In *Neural Information Processing Systems (NeurIPS)*, 2022.

Takakura, S. and Suzuki, T. Approximation and estimation ability of transformers for sequence-to-sequence functions with infinite dimensional input. In *International Conference on Machine Learning*, pp. 33416–33447. PMLR, 2023.

Taylor, R., Kardas, M., Cucurull, G., Scialom, T., Hartshorn, A., Saravia, E., Poulton, A., Kerkez, V., and Stojnic, R. Galactica: A Large Language Model for Science, November 2022.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. Transformers learn in-context by gradient descent, May 2023.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. Emergent Abilities of Large Language Models, October 2022.

Weiss, G., Goldberg, Y., and Yahav, E. Thinking Like Transformers, July 2021.

Wu, J., Zou, D., Chen, Z., Braverman, V., Gu, Q., and Bartlett, P. L. How many pretraining tasks are needed for in-context learning of linear regression? *arXiv preprint arXiv:2310.08391*, 2023.

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*, 2021.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

Yun, C., Bhojanapalli, S., Rawat, A. S., Reddi, S. J., and Kumar, S. Are Transformers universal approximators of sequence-to-sequence functions?, February 2020.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Sy8gdB9xx.

Zhang, R., Frei, S., and Bartlett, P. L. Trained Transformers Learn Linear Models In-Context, October 2023.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J., Bengio, S., and Nakkiran, P. What algorithms can transformers learn? a study in length generalization. *arXiv preprint arXiv:2310.16028*, 2023.

Zou, A., Phan, L., Chen, S., Campbell, J., Guo, P., Ren, R., Pan, A., Yin, X., Mazeika, M., Dombrowski, A.-K., et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*, 2023.

Organization of the Appendix

A. Examples for Different Tasks

Here we use letters `a,b,...` to denote the "question tokens" (also can be seen as pixel blocks for input images in Figure 1, use integer number `0,1,2,...` denote the labels, such as "hat" and "car", and use → denote the response sign (or `CLS`), then we provide data examples as in Table 1, for each example, the prior part of the sequence is the input, and the model should predict the underlined result:

Table 1: Examples of four tasks.

Task	Data Example	Explanation
Memorization	aa→1 bb→2 cc→3 ab→4 ac→5 ba→6 bc→7 ca→8 cb→9	different sequence belong to different class
Reasoning	a→1 b→2 a→1 b→1 a→2 a→2 a→1 b→2 b→2 b→1 a→2 b→1	answer the last question based on the context
Generalization	aa→1 bb→1 cc→1 ab→2 ac→2 ba→2 bc→2 ca→2 cb→2	sequence generated by same template belongs to same class different template have different label (in this example we have templates $\alpha\alpha\to 1$, $\alpha\beta\to 2$)
Contextual Generalization	aa→1 ab→2 bb→1 aa→1 ab→2 ba→2 aa→1 ab→2 aa→1 aa→1 ab→2 ab→2	similar question have the same answer answer the last question based on the context

Memorization Task: We assume that each sequence belongs to a different cluster, meaning that the sequences are "independent" from each other. This means that the model only needs to memorize each sequence and its corresponding label without considering any relationships between the data.

Reasoning Task: We first provide question-answer pairs and then ask the model to retrieve the answer from the context. Such task can be used as a benchmark to evaluate the model's reasoning and comprehension ability (Liu et al., 2023).

Generalization Task: we assume that sequences generated by the same template belong to the same class. For example, different-colored cars should be classified into the same class. Our setting is adapted from (Boix-Adsera et al., 2023), and we focus on understanding when and how the model can generalize to **all possible** sequences that belong to the same template.

Contextual Generalization: We assume that similar questions (generated by the same template) should have the same answer. Therefore, the task requires analyzing the semantic similarity between each question and retrieving the answer from the context.

Reasoning focuses on the semantic meaning behind the sequence, while generalization focuses on the relationships between each sequence. This makes our task more challenging compared to the memorization task, which ignores any possible relationships within and between the data.

B. Supporting Experiments

B.1. Experiments setup

Model: Our model is an attention-only transformer with a classification layer. We employ different initialization methods for different tasks. For the memorization task, we initialize $\mathbf{W}_{QK}$ with a uniform distribution in the range $[0, 1)$. For other tasks, we initialize $\mathbf{W}_{QK}$ in the first layer of the transformer with constructed pattern for different tasks, as illustrated in Appendix F.2, Appendix G.2, and Appendix H. This allows us to provide a favorable starting point for these models. Additionally, we initialize $\mathbf{W}_V$ for each task with a value of 1. The classification layer $\mathbf{W}_C$ is initialized with a uniform distribution in the range $[0, 1)$. We have observed that tasks related to generalization and initialization can be challenging even for randomly initialized transformers with sufficient layers. Our objective is to gain insights into the workings of transformers on these tasks and empirically validate the propositions in Section 5. Hence, this choice of initialization is justified.

Data: We construct the dataset as described in Section 3. Before inputting the data to the transformer, we concatenate each token with a one-hot positional encoding. This is done to ensure that different types of tokens are disentangled, meaning that each token is encoded in a certain subspace. For memorization tasks, we set the word dimension d_1 to 500 and add an extra dimension for the response sign, resulting in a final token dimension of $d = 500 + 1$. The sequence length is set to 6, so the one-hot positional encoding has a dimension of $n = 6 + 1$. We randomly choose e examples and assign them with random permuted e distinct labels. Therefore, the input sequence is represented as a $(d + n) \times n$ matrix, where the transformer hidden size is set to $d' = (d + n) = 508$.

Similarly, for reasoning tasks, we set the question dimension d_1 to 100, the answer dimension d_2 to 100, and include an extra dimension for the response sign. This results in a token dimension of $d = 100 + 100 + 1$. For a sequence with k examples, the total length is $n = 3k + 2$.

For generalization tasks, we set the word dimension to $d_1 = 210$ and add an extra dimension for the response sign. The input sequence length is $n = l + 1$, where l represents the template length.

For contextual generalization tasks, we set the question dimension $d_1 = 100$, the answer dimension $d_2 = 100$, and each question is generated from a template length $l = 5$. For a sequence with k examples, the total length is $n = (l + 1 + 1)(k + 1) - 1 = 7k + 6$.

Training: During training, we utilize stochastic gradient descent (SGD) as the optimizer with cross-entropy loss function given by:

$$\ell(y, \mathbf{o}) = -\log \frac{e^{\mathbf{o}[y]}}{\sum e^{\mathbf{o}[i]}}$$

Here, $\hat{\mathbf{o}}$ represents the prediction result based on the last response sign, and y represents the target index. In the case of memorization and template generalization tasks, y can be viewed as a label. For reasoning and contextual reasoning tasks, y corresponds to a vocabulary index.

B.2. The Impact of the Number of Attention Heads

In our previous experiments, we discovered that a single-head transformer with 2 layers is sufficient for performing reasoning and generalization tasks. However, in practice, it is common to use multiple attention heads, so we conducted additional experiments to investigate the effects of using more than one attention head. In the following figure, we represent the number of heads on each layer using the list $[h_1, h_2, \cdots]$.

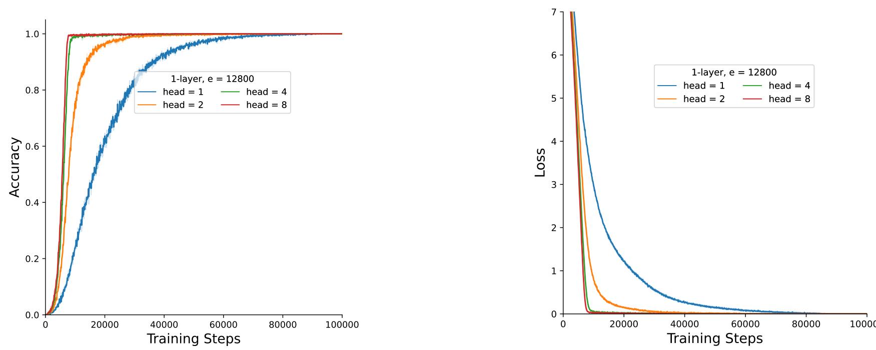

Figure 5: Training dynamic for different attention heads on memorization task

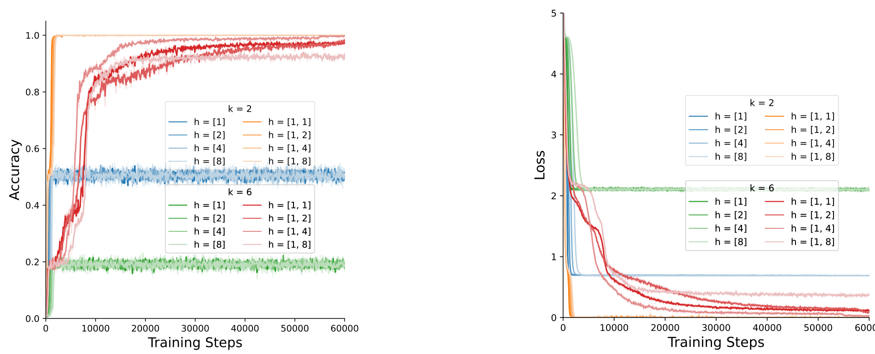

Figure 6: Training dynamic for different attention heads on in-context learning task

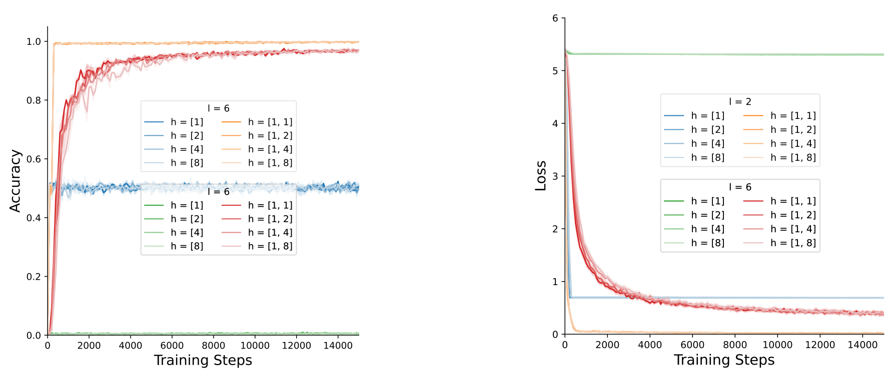

Figure 7: Training dynamic for different attention heads on template generalization task

For the memorization task, increasing the number of attention heads allows the model to more easily memorize the data, as shown in Figure 5. However, for tasks that require reasoning and generalization, such as those depicted in Figure 6 and Figure 7, additional attention heads does not yield a significant impact.

B.3. Additional experiments for contextual generalization task

When it comes to more challenging contextual generalization task, we conduct extensive additional experiments to investigate the impact of both the number of layer and attention heads.

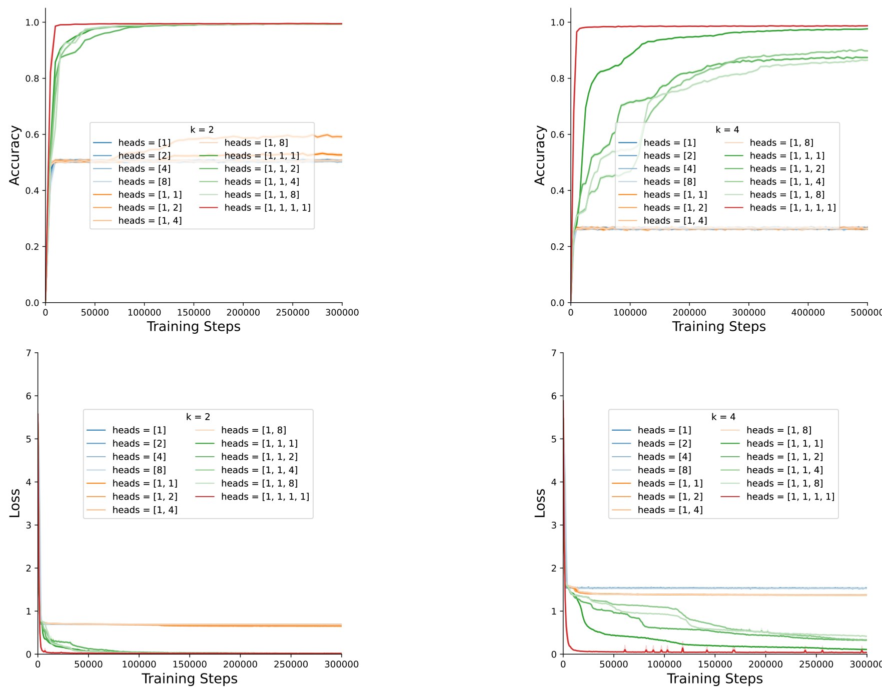

Figure 8: Training dynamic for different attention heads and layers on contextual generalization task on given k examples

As shown in Figure 8, similar to the results in Figure 6 and Figure 7, the addition of attention heads does not have a significant effect for generalization and reasoning. It is worth noting that when each sequence contains 4 examples, a 3-layer transformer with a single head in the last layer exhibits faster learning compared to models with multiple attention heads. This can be attributed to the inherent nature of larger models, which typically require more steps to converge (Bottou et al., 2018). Interestingly, we observed that a 4-layer transformer with a single head achieves even faster learning in the contextual generalization task compared to the 3-layer transformer. This suggests that by increasing the number of layers, the model can employ more complex and efficient methods to solve problems.

C. Useful Transformer Constructions

In this section, we introduce two useful constructions that can help us understand the linear attention model and make the following section of our construction clearer. The *instructive attention* enables us to generate any token-invariant attention map $\boldsymbol{\alpha}$, i.e., the attention map $\boldsymbol{\alpha} \in \mathbb{R}_+^{n\times n}$ is independent of the token embedding, but instead only relies on the position in the sequence. This implies that we can instruct the model to focus on the specific areas for each position. The *constrained attention* allows us to apply custom masks to the original attention, restricting the attention of each token to a specific segment of the sequence. This will be applied as the core to make transformer generalization more effective.

Lemma C.1 (Instructive attention)**.** *There exists an attention layer that can guide the attention between each token using the positional encoding. Let* $\mathbf{H} \in \mathbb{R}^{d'\times n}$ *be the input, represented as:*

$$\mathbf{H} = \begin{pmatrix} \mathbf{X} \\ \mathbf{P} \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_0 & \cdots & \mathbf{x}_{n-1} \\ \mathbf{p}_0 & \cdots & \mathbf{p}_{n-1} \\ \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} \begin{matrix} \to d\times n \\ \to n \times n \\ \to (d'-n)\times n \end{matrix}.$$

For any attention score $\boldsymbol{\alpha} \in \mathbb{R}_+^{n\times n}$ *that is independent of the input sequences, there always exists a weight matrix* $\mathbf{W}_{QK} \in \mathbb{R}^{d'\times d'}$ *such that the following holds for any input sequence* $(\mathbf{x}_0, \cdots, \mathbf{x}_{n-1}) \in \mathcal{X}^n$*:*

$$\sigma\left(\mathbf{H}^\top \mathbf{W}_{QK}\mathbf{H}\right) = \boldsymbol{\alpha}$$

Proof. Recall that the sequence will first be preprocessed into a $d' \times n$ matrix with positional encoding, and padding zero to make the input dimension align with the model's hidden size d', result in the final input $\mathbf{H}$. Besides, note that we consider the one-hot positional encoding, i.e., $\mathbf{p}_i = [\underbrace{0,\ldots,0}_{i}, 1, \underbrace{0,\ldots,0}_{n-(i+1)}]^\top$. Then, we can first consider the attention weight matrix $\mathbf{W}_{QK}$ with the following form:

$$\mathbf{W}_{QK} = \begin{pmatrix} \mathbf{W}_{tt} & \mathbf{W}_{tp} & \mathbf{0}_{d\times(d'-n)} \\ \mathbf{W}_{pt} & \mathbf{W}_{pp} & \mathbf{0}_{n\times(d'-n)} \\ \mathbf{0}_{(d'-n)\times d} & \mathbf{0}_{(d'-n)\times n} & \mathbf{0}_{(d'-n)\times(d'-n)} \end{pmatrix}.$$

Expanding the equation $\mathbf{H}^\top \mathbf{W}_{QK}\mathbf{H}$ gives:

$$\mathbf{H}^\top \mathbf{W}_{QK}\mathbf{H} = \mathbf{X}^\top \mathbf{W}_{tt}\mathbf{X} + \mathbf{P}^\top \mathbf{W}_{pt}\mathbf{X} + \mathbf{X}^\top \mathbf{W}_{tp}\mathbf{P} + \mathbf{P}^\top \mathbf{W}_{pp}\mathbf{P}. \tag{4}$$

Note that $\mathbf{P} = [\mathbf{p}_0, \cdots, \mathbf{p}_{n-1}] = \mathbf{I}$. Therefore, we can further set $\mathbf{W}_{tt} = \mathbf{0}$, $\mathbf{W}_{pt} = \mathbf{0}$, and $\mathbf{W}_{tp} = \mathbf{0}$, then

$$\sigma\left(\mathbf{H}^\top \mathbf{W}_{QK}\mathbf{H}\right) = \sigma\left(\mathbf{P}^\top \mathbf{W}_{pp}\mathbf{P}\right) = \sigma\left(\mathbf{W}_{pp}\right).$$

It is then clear that the attention map only depends on $\mathbf{W}_{pp}$ and has nothing to do with the token embeddings $\mathbf{X}$. Therefore, we can directly set $\mathbf{W}_{pp} = \boldsymbol{\alpha}$ to complete the proof. □

Lemma C.1 shows that the transformer can instruct the attention through positional encoding. That is, the transformer can aggregate any tokens together for further processing, which is essential for the instruction head where multiple tokens need to be copied together for further processing (Olsson et al., 2022).

Furthermore, applying the above construction can further help implement attention masking, which aims to restrict attention between any specific pair of tokens based on their positions:

Lemma C.2 (Constrained attention)**.** *There exists an attention layer that can control the attention between any pair of tokens through the one-hot positional encoding, i.e., masking the correlation of tokens between any two positions.*

Proof. By Equation (4), we have

$$\mathbf{H}^\top \mathbf{W}_{QK}\mathbf{H} = (\mathbf{X}^\top \mathbf{W}_{tt}\mathbf{X} + \mathbf{P}^\top \mathbf{W}_{pt}\mathbf{X} + \mathbf{X}^\top \mathbf{W}_{tp}\mathbf{P}) + \mathbf{W}_{pp}.$$

The first term on the R.H.S. of the above equation enables the model to learn attention between each token at different positions, while the right part $\mathbf{W}_{pp}$ can be arbitrarily designed. Therefore, in order to mask the attention between the tokens at positions i and j, we can directly set $\mathbf{W}_{pp}(i,j) \to \infty$ or some sufficiently large negative value. In this way, we can constrain the attention between each token based on their position. □

In the following sections, these two attention methods will be useful to help achieve the desired task. In particular, instructive attention to aggregate tokens at specific positions, and utilize the constrained attention mechanism when the model needs to concentrate on a particular segment of the sequence.

D. Limitation for Single Layer Attention-only Transformer

In this section, we present an intriguing scenario where the input sequences possess specific properties. Under these conditions, the predictions made by a single layer transformer exhibit **linear dependence**, implying that the sequence labels must satisfy specific constraints. This limitation hinders the performance of the 1-layer transformer, and we will leverage this property to demonstrate why the single layer transformer struggles with reasoning and generalization in Appendix F.1 and Appendix G.1.

To begin, let us define a combination operation for discrete tokens and sequences, which is useful in describing the relationship among sequences.

Definition D.1. (*Combination operation for discrete tokens and sequences*). Let $\mathbf{x}$ and $\mathbf{y}$ be two tokens chosen from the vocabulary $\mathcal{X}$. The combination operation for these two tokens, denoted as $\mathbf{x} \oplus \mathbf{y}$, is a multiset containing both tokens, i.e., $\mathbf{x} \oplus \mathbf{y} = [\mathbf{x}, \mathbf{y}]$. Furthermore, the combination between a multiset $[\mathbf{x}, \mathbf{y}]$ and a token $\mathbf{z}$ results in a new multiset $[\mathbf{x}, \mathbf{y}, \mathbf{z}]$. For two sequences $\mathbf{X} = (\mathbf{x}_0, \dots, \mathbf{x}_{n-1})$ and $\mathbf{Y} = (\mathbf{y}_0, \dots, \mathbf{y}_{n-1})$, their combination $\mathbf{X} \oplus \mathbf{Y} = (\mathbf{x}_0 \oplus \mathbf{y}_0, \dots, \mathbf{x}_{n-1} \oplus \mathbf{y}_{n-1})$ is a sequence of multisets. Notably, for any positive integer $\lambda \in \mathbb{Z}_+$, $\lambda \otimes \mathbf{x}$ can be interpreted as a multiset consisting of λ copies of token $\mathbf{x}$, i.e., $\{\underbrace{\mathbf{x}, \dots, \mathbf{x}}_{k \text{ times}}\}$. Similarly, $\lambda \otimes \mathbf{X} = (\lambda \otimes \mathbf{x}_0, \dots, \lambda \otimes \mathbf{x}_{n-1})$.

Definition D.2. (*Sequence dependent*). We define the input sequences $\mathbf{X}^{(0)}, \mathbf{X}^{(1)}, \dots, \mathbf{X}^{(N-1)} \in \mathcal{X}^n$ as *dependent* if they satisfy the following conditions:

1. All sequences end with the same token, i.e., $\mathbf{X}^{(0)}[n-1] = \mathbf{X}^{(1)}[n-1] = \cdots = \mathbf{X}^{(N-1)}[n-1]$.

2. There exist coefficients $\lambda_0, \lambda_1, \dots, \lambda_{N-1} \in \mathbb{Z}$ (not all zero) such that

$$\sqcap_{i \in \mathcal{I}_+} (\lambda_i \otimes \mathbf{X}^{(i)}) = \sqcap_{i \in \mathcal{I}_-} (-\lambda_i \otimes \mathbf{X}^{(i)}), \tag{5}$$

   where $\mathcal{I}_+ = \{i | \lambda_i \geq 0, i \in [N]\}$ and $\mathcal{I}_- = \{i | \lambda_i < 0, i \in [N]\}$ divide the sequences into two parts. Here, $\sqcap_{i \in \mathcal{I}} \lambda_i \otimes \mathbf{X}^{(i)} = (\lambda_{i_1} \otimes \mathbf{X}^{(i_1)}) \oplus \cdots \oplus (\lambda_{i_k} \otimes \mathbf{X}^{(i_k)})$ represents the operation of combining sequences.

Example Consider a vocabulary $\mathcal{X} = \{\mathbf{a}, \mathbf{b}, \mathbf{r}\}$ with 3 discrete tokens. In this case, the sequences $(\mathbf{a}, \mathbf{b}, \mathbf{r}), (\mathbf{b}, \mathbf{a}, \mathbf{r}), (\mathbf{a}, \mathbf{a}, \mathbf{r}), (\mathbf{b}, \mathbf{b}, \mathbf{r})$ are dependent. This is because $(\mathbf{a}, \mathbf{b}, \mathbf{r}) \oplus (\mathbf{b}, \mathbf{a}, \mathbf{r}) = ([\mathbf{a}, \mathbf{b}], [\mathbf{a}, \mathbf{b}], [\mathbf{r}, \mathbf{r}]) = (\mathbf{a}, \mathbf{a}, \mathbf{r}) \oplus (\mathbf{b}, \mathbf{b}, \mathbf{r})$. Additionally, we can observe that $\sum_{i=0}^{N-1} \lambda_i = 0$ since both sides of Equation (5) should have the same number of discrete tokens in each position.

Proposition D.3. *If the input sequences $\mathbf{X}^{(0)}, \mathbf{X}^{(1)}, \dots, \mathbf{X}^{(N-1)} \in \mathcal{X}^n$ are **dependent**, then for the first attention layer $\mathtt{TF}_1$ for any attention only transformer, we have*

$$\lambda_0(\mathtt{TF}_1(\mathbf{X}^{(0)})[n-1]) + \lambda_1(\mathtt{TF}_1(\mathbf{X}^{(1)})[n-1]) + \cdots + \lambda_{N-1}(\mathtt{TF}_1(\mathbf{X}^{(N-1)})[n-1]) = \mathbf{0}.$$

Here, $\{\lambda_i\}_{i=0}^{N-1}$ represents the coefficients defined in Definition D.2. We highlight the linear dependency in the output of at position $n-1$, which can be regarded as the representation for the next token or classification prediction of the entire sequence.

Proof. Assuming the input sequence is $\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1}] \in \mathbb{R}^{d \times n}$, the 1-layer linear attention only transformer $\mathtt{TF}_1$ performs a sequence-to-sequence mapping by first computing the attention score between each token and then aggregating them to obtain the final representation at each position:

$$\begin{aligned} \mathtt{TF}_1(\mathbf{X}) &= \mathbf{X} + \frac{1}{m} \sum_{i=1}^{m} \left[(\mathbf{W}_{V_i} \mathbf{X}) \sigma((\mathbf{W}_{Q_i} \mathbf{X})^\top (\mathbf{W}_{K_i} \mathbf{X})) \right] \\ &= \mathbf{X} + \frac{1}{m} \sum_{i=1}^{m} \left[(\mathbf{W}_{V_i} \mathbf{X}) \sigma(\mathbf{X}^\top \mathbf{W}_{QK_i} \mathbf{X}) \right] \end{aligned} \tag{6}$$

Here, we have combined the learnable parameters $\mathbf{W}_{Q_i}, \mathbf{W}_{K_i} \in \mathbb{R}^{h \times d}$ into a single matrix $\mathbf{W}_{QK_i} \in \mathbb{R}^{d \times d}$ for analysis simplification.

Consider the output at position $k \in [n]$:

$$
\begin{aligned}
\mathtt{TF}_1(\mathbf{X})[k] &= \mathbf{x}_k + \frac{1}{m}\sum_{i=1}^{m}\left[(\mathbf{W}_{V_i}\mathbf{X})\sigma(\mathbf{X}^\top \mathbf{W}_{QK_i}\mathbf{x}_k)\right] \\
&= \mathbf{x}_k + \frac{1}{m}\sum_{i=1}^{m}\sum_{j=0}^{n-1}\left[(\mathbf{W}_{V_i}\mathbf{x}_j)\sigma(\mathbf{x}_j^\top \mathbf{W}_{QK_i}\mathbf{x}_k)\right] \\
&= \mathbf{x}_k + \frac{1}{m}\sum_{j=0}^{n-1}\sum_{i=1}^{m}\left[(\mathbf{W}_{V_i}\mathbf{x}_j)\sigma(\mathbf{x}_j^\top \mathbf{W}_{QK_i}\mathbf{x}_k)\right],
\end{aligned} \tag{7}
$$

we define $\mathtt{Attn}(\mathbf{x}_j, \mathbf{x}_k) := \frac{1}{m}\sum_{i=1}^{m}\left[(\mathbf{W}_{V_i}\mathbf{x}_j)\sigma(\mathbf{x}_j^\top \mathbf{W}_{QK_i}\mathbf{x}_k)\right]$, in this way the output at position k can be written as $\mathtt{TF}(\mathbf{X})[k] = \mathbf{x}_k + \sum_{j=1}^{n}\mathtt{Attn}(\mathbf{x}_j, \mathbf{x}_k)$. Specifically, when the last token for all input sequences is the same, the prediction based on the last token is:

$$
\mathtt{TF}_1(\mathbf{X})[n-1] = \mathbf{x}_{n-1} + \sum_{j=0}^{n-1}\mathtt{Attn}(\mathbf{x}_j, \mathbf{x}_{n-1}). \tag{8}
$$

If the input sequences are dependent, based on Definition D.2, we can divide the sequences into two groups $\mathcal{I}_+$ and $\mathcal{I}_-$ so in each position j, both side have the same occurrence for each token:

$$
\sqcap_{i\in\mathcal{I}_+}(\lambda_i \otimes \mathbf{x}_j^{(i)}) = \sqcap_{i\in\mathcal{I}_-}(-\lambda_i \otimes \mathbf{x}_j^{(i)}) := \mathcal{S}_j, \tag{9}
$$

here we use $\mathcal{S}_j$ to denote the tokens occurrences at position j, note that $\lambda\mathtt{Attn}(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1}) = \sum_{\mathbf{s}\in(\lambda\otimes\mathbf{x}_j^{(i)})}\mathtt{Attn}(\mathbf{s}, \mathbf{x}_{n-1})$, so we can derive the following equation:

$$
\sum_{i\in\mathcal{I}_+}\lambda_i\mathtt{Attn}(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1}) = \sum_{\mathbf{s}\in\mathcal{S}_j}\mathtt{Attn}(\mathbf{s}, \mathbf{x}_{n-1}) = \sum_{i\in\mathcal{I}_-}-\lambda_i\mathtt{Attn}(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1}). \tag{10}
$$

Then for the final prediction result

$$
\begin{aligned}
&\lambda_0(\mathtt{TF}_1(\mathbf{X}^{(0)})[n-1]) + \lambda_1(\mathtt{TF}_1(\mathbf{X}^{(1)})[n-1]) + \cdots + \lambda_{N-1}(\mathtt{TF}_1(\mathbf{X}^{(N-1)})[n-1]) \\
=&\left(\mathbf{x}_{n-1}\sum_{i=0}^{N-1}\lambda_i\right) + \left(\sum_{j=0}^{n-1}\sum_{i=0}^{N-1}\lambda_i\mathtt{Attn}(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1})\right) \\
=&\mathbf{0}.
\end{aligned} \tag{11}
$$

The first part of Equation (11) is zero based on the observation that $\sum_{i=0}^{N-1}\lambda_i = 0$ since both sides of Equation (5) should have the same number of discrete tokens in each position, and the second part is zero based on Equation (10), which states that each token should have the same occurrence for both groups. □

In a single layer transformer, which consists of only one attention layer and a linear classifier layer, the result $\mathbf{o} = \mathbf{W}_O\mathtt{TF}_1(\mathbf{X})[n-1])$ also satisfies the linear dependency:

Proposition D.4. *If the input sequences $\mathbf{X}^{(0)}, \mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N-1)} \in \mathcal{X}^n$ are **dependent**, then for any single layer transformer $\mathtt{TF}$, their prediction result $\mathbf{o}^{(0)}, \ldots, \mathbf{o}^{(N-1)}$*

$$
\lambda_0\mathbf{o}^{(0)} + \lambda_1\mathbf{o}^{(1)} + \cdots + \lambda_{N-1}\mathbf{o}^{(N-1)} = \mathbf{0}.
$$

Here, $\{\lambda_i\}_{i=0}^{N-1}$ represents the coefficients defined in Definition D.2.

Proof. By directly applying Proposition D.3 and multiplying both sides by $\mathbf{W}_O$, we have:

$$\lambda_0 \mathbf{W}_O(\texttt{TF}_1(\mathbf{X}^{(0)})[n-1]) + \lambda_1 \mathbf{W}_O(\texttt{TF}_1(\mathbf{X}^{(1)})[n-1]) + \cdots + \lambda_{N-1} \mathbf{W}_O(\texttt{TF}_1(\mathbf{X}^{(N-1)})[n-1]) = \mathbf{W}_O\mathbf{0}.$$

In a single layer transformer, the prediction is conducted by a linear projection $\mathbf{W}_O \in \mathbb{R}^{c\times d'}$. Therefore, we can replace $\mathbf{W}_O\texttt{TF}_1(\mathbf{X}^{(i)}[n-1])$ with $\mathbf{o}^{(i)}$, thus completing the proof. □

According to Proposition D.4, the prediction of a single layer transformer for certain dependent sequences should be linearly dependent. For instance, let's consider the four sequences: $(\mathbf{a}, \mathbf{b}, \mathbf{r}), (\mathbf{b}, \mathbf{a}, \mathbf{r}), (\mathbf{a}, \mathbf{a}, \mathbf{r}), (\mathbf{b}, \mathbf{b}, \mathbf{r})$. In this case, the final predictions $\mathbf{o}_0, \mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3$ will satisfy the equation $\mathbf{o}_0 + \mathbf{o}_1 = \mathbf{o}_2 + \mathbf{o}_3$ for any single layer transformer. This observation implies that the labels of these sequences are restricted for a single layer transformer to fit all of them. We will explain later in Appendix F.1 and Appendix G.1 why this leads to the failure of the single layer transformer in reasoning and generalization. Besides, we extend Proposition D.4 to from ReLU attention to softmax attention for single layer single head transformer in Appendix I.

E. Proofs for Section 5.1

In this section, we will prove the results in Section 5.1, which asserts that a single-layer transformer with a sufficient number of attention heads can effectively memorize sequences with distinct labels. We establish this by first proving the existence of a linear projection layer that can map any non-parallel vectors with distinct labels. Subsequently, we provide evidence that with a sufficient number of attention heads, there exists a transformer can map any sequences to non-parallel vectors.

In the following section, we will first show that there exists a single-layer transformer with n heads, $d' = \max\{nd, d+n\}$, that any input sequence $\mathbf{H} \in \mathbb{R}^{d'\times n}$ can be mapped to a corresponding distinct label so that the memorizing task can be perfectly performed, our construction procedure can be formulated as follows:

$$\mathbf{H} = \begin{pmatrix} \overbrace{\begin{matrix}\mathbf{x}_0 & \cdots & \mathbf{x}_{k-1}\end{matrix}}^{\substack{sequence \\ \text{with } k \text{ tokens}}} & \overbrace{\mathbf{0}}^{\substack{response \\ sign}} \\ \begin{matrix}0 & \cdots & 0\end{matrix} & 1 \\ \begin{matrix}\mathbf{p}_0 & \cdots & \mathbf{p}_{k-1}\end{matrix} & \mathbf{p}_k \\ \begin{matrix}\mathbf{0} & \cdots & \mathbf{0}\end{matrix} & \mathbf{0} \end{pmatrix} \xrightarrow[Lemma\ E.2]{\texttt{TF}\text{, last token}} \begin{pmatrix} \mathbf{x}_0 \\ \vdots \\ \mathbf{x}_{k-1} \\ \mathbf{0} \end{pmatrix} \xrightarrow[Lemma\ E.1]{\text{linear classifier}} \begin{pmatrix} o_0 \\ \vdots \\ o_j \\ \vdots \\ o_{C-1} \end{pmatrix} \begin{matrix}\text{max element} \\ (\text{distinct } j \text{ for each sequence})\end{matrix}$$

We first deliver the following lemma, which proves that the linear classify can perfectly map the representation of any input sequence to the desired labels.

Lemma E.1. *Let $\{(\mathbf{x}_0, y_0), \ldots, (\mathbf{x}_{N-1}, y_{N-1})\} \subset \mathbb{R}^{d\times N} \times [N]^N$ be a dataset of N vector-label pairs, where every two vectors $\mathbf{x}_i, \mathbf{x}_j$ are linearly independent, and the labels of each vector are different. Then, there exists a linear projector $\mathbf{W} \in \mathbb{R}^{d\times N}$ such that:*

$$(\mathbf{x}_i\mathbf{W})[y_i] > (\mathbf{x}_i\mathbf{W})[j] \quad \forall j \in [N], j \neq y_i, \forall i \in [N].$$

Proof. Considering the result of the linear projector as the prediction of the probability of the sequence belonging to each class $1, \ldots, N$, it always predicts the highest possibility at the index of the corresponding label. In this case, the model will achieve a perfect accuracy rate. We construct $\mathbf{W} = [\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{N-1}]$ by assigning the value $\mathbf{w}$ at each column one by one. WLOG, we let the data be $\{(\mathbf{x}_0, 0), (\mathbf{x}_1, 1), \ldots, (\mathbf{x}_{N-1}, N-1)\}$.

We started by consider the case $N = 2$, because $\mathbf{x}_0, \mathbf{x}_1$ are linearly independent, for any $\mathbf{a}, \mathbf{b} \in \mathbb{R}^2$, Equation (12) have at least one solution for $\begin{bmatrix}\mathbf{w}_0 & \mathbf{w}_1\end{bmatrix}$.

$$\begin{pmatrix}\mathbf{x}_0 \\ \mathbf{x}_1\end{pmatrix} \begin{pmatrix}\mathbf{w}_0 & \mathbf{w}_1\end{pmatrix} = \begin{pmatrix}\mathbf{a} \\ \mathbf{b}\end{pmatrix}, \tag{12}$$

we can let $\mathbf{a} = [1, 0], \mathbf{b} = [0, 1]$ and solve the above equation, the corresponding solution $\mathbf{w}_0, \mathbf{w}_1$ can correctly classify $\{(\mathbf{x}_0, 0), (\mathbf{x}_1, 1)\}$.

If $N = k$ satisfies the above condition, which means that there exists $[\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_{k-1}]$ that can correctly classify $\{(\mathbf{x}_0, 0), (\mathbf{x}_1, 1), \dots, (\mathbf{x}_{k-1}, k-1)\}$, then consider a new data point $(\mathbf{x}_k, k)$. We construct $\mathbf{w}_k$ as follows:

First, we compute the result $\mathbf{o}^{(k)}_{:k} = \mathbf{x}_k[\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_{k-1}]$ and choose the highest index $i_k = \text{argmax}_i \mathbf{o}^{(k)}_{:k}$. Then, we consider the result if we set $\mathbf{w}_k = \mathbf{w}_{i_k}$:

$$\begin{pmatrix} \mathbf{x}_0 \\ \vdots \\ \mathbf{x}_{i_k} \\ \vdots \\ \mathbf{x}_k \end{pmatrix} \Big(\overbrace{\mathbf{w}_0 \quad \dots \quad \mathbf{w}_{i_k} \quad \dots \quad \mathbf{w}_{k-1}}^{k} \quad \underbrace{\mathbf{w}_{i_k}}_{\substack{\text{init} \\ \mathbf{w}_k = \mathbf{w}_{i_k}}} \Big) = \begin{pmatrix} o_{o,o} & \cdots & o_{0,i_k} & \cdots & o_{0,k-1} & o_{0,i_k} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ o_{i_k,0} & \cdots & o_{i_k,i_k} & \cdots & o_{i_k,k-1} & o_{i_k,i_k} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ o_{k-1,0} & \cdots & o_{k-1,i_k} & \cdots & o_{k-1,k-1} & o_{k-1,i_k} \\ o_{k,0} & \cdots & o_{k,i_k} & \cdots & o_{k,k-1} & o_{k,i_k} \end{pmatrix},$$

We can see that appending $\mathbf{w}_{i_k}$ wouldn't change the prediction result for $\{\mathbf{x}_0, \dots, \mathbf{x}_{i_k-1}, \mathbf{x}_{i_k+1}, \dots, \mathbf{x}_{k-1}\}$. Therefore, we just need to find an extra $\Delta\mathbf{w}_k$ such that $\mathbf{w}_k = \mathbf{w}_{i_k} + \Delta\mathbf{w}_k$ remains the prediction for $\{\mathbf{x}_0, \dots, \mathbf{x}_{i_k-1}, \mathbf{x}_{i_k+1}, \dots, \mathbf{x}_k\}$ unchanged (Equation (13)), and can successfully classify $(\mathbf{x}_{i_k}, i_k)$ and $(\mathbf{x}_k, k)$ (Equation (14)).

$$\begin{pmatrix} \mathbf{x}_0 & \dots & \mathbf{x}_{i_k-1} & \mathbf{x}_{i_k+1} & \cdots & \mathbf{x}_{k-1} \end{pmatrix}^\top \Delta\mathbf{w}_k \prec \begin{pmatrix} \epsilon & \dots & \epsilon & \epsilon & \cdots & \epsilon \end{pmatrix}^\top, \tag{13}$$

$$\begin{cases} \mathbf{x}_{i_k}\Delta\mathbf{w}_k & < 0 \\ \mathbf{x}_k\Delta\mathbf{w}_k & > 0 \end{cases}, \tag{14}$$

where $\epsilon = \min_{i \in [k]_{\dagger i_k}} \{p_{i,i} - p_{i,i_k}\}$, which is also a maximum vibration that can remain the prediction result for $\{\mathbf{x}_0, \dots, \mathbf{x}_{i_k-1}, \mathbf{x}_{i_k+1}, \dots, \mathbf{x}_{k-1}\}$ unchanged. Such $\Delta\mathbf{w}_k$ exists as $\mathbf{x}_{i_k}$ and $\mathbf{x}_k$ are linearly independent. Therefore, we can first solve the equation

$$\begin{cases} \mathbf{x}_{i_k}\Delta\mathbf{w}_k & = -1 \\ \mathbf{x}_k\Delta\mathbf{w}_k & = 1 \end{cases}, \tag{15}$$

and then rescale the $\Delta\mathbf{w}_k = \frac{1}{M}\Delta\mathbf{w}_k$ to ensure that the vibration for any other rows is less than ϵ. Finally, we assign $\mathbf{w}_k = \Delta\mathbf{w}_k + \mathbf{w}_{i_k}$, which ensures that the prediction result for $\mathbf{x}_k$ is k, while the prediction for other rows remains unchanged.

□

Given the mapping capability of sequences to non-parallel vectors using Lemma E.1, it follows that constructing a classifier layer to map each vector to a distinct label becomes straightforward. Consequently, our focus turns to the second part, where we aim to establish the validity of Lemma E.2. This lemma asserts that a single-layer transformer, equipped with n attention heads and a hidden size of $\max\{kd, d+k\}$, can effectively map a sequence to a non-parallel vector.

Lemma E.2. *Given a vocabulary $\mathcal{X}$ of non-parallel vectors, then there exists a single-layer transformer with n attention heads and hidden size $d' = \max\{k \cdot d, d+k\}$ to map all possible sequences of length n (ending with a response sign) to non-parallel vectors.*

Proof. To achieve this, we construct a transformer with n attention heads, where each head processes only the token at its corresponding position i, for the first k heads, we process the sequence for the last token (response sign) as follows:

$$\mathbf{W}_{V_i} \begin{pmatrix} \mathbf{x}_0 & \cdots & \mathbf{x}_{k-1} & \mathbf{0} \\ 0 & \cdots & 0 & 1 \\ \mathbf{p}_0 & \cdots & \mathbf{p}_{k-1} & \mathbf{p}_k \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix}, \quad \boldsymbol{\alpha}_i[:, n-1] = n \begin{pmatrix} \mathbf{0}_{i \cdot d \times 1} \\ \mathbf{x}_i \\ \mathbf{0}_{(d'-(i+1)d) \times 1} \end{pmatrix},$$

where $\boldsymbol{\alpha}_i \in \mathbb{R}^{n\times n}$ is the attention map for the i-th head, we can assign $\boldsymbol{\alpha}_i[:, n-1] = [\underbrace{0,\dots,0}_{i}, 1, \underbrace{0,\dots,0}_{n-(i+1)}]^\top$ using the construction for instructive attention Lemma C.1, and we set $\mathbf{W}_{V_i}$ to copy the token at position i in a disentangled manner:

$$\mathbf{W}_{V_i} = n\begin{pmatrix} \mathbf{0}_{i\cdot d\times d} & \mathbf{0}_{i\cdot d\times(d'-d)} \\ \mathbf{I}_{d\times d} & \mathbf{0}_{d\times(d'-d)} \\ \mathbf{0}_{(d'-(i+1)d)\times d} & \mathbf{0}_{(d'-(i+1)d)\times(d'-d)} \end{pmatrix}.$$

Then we set the k-th head to neutralize the information from the residue:

$$\mathbf{W}_{V_k} = -n\mathbf{I} \quad \boldsymbol{\alpha}_k = \mathbf{I}.$$

As a result, the processed token representation at the last position is:

$$\begin{aligned} \mathbf{H}^{(1)}[n-1] &= \mathbf{H}^{(0)}[n-1] + \frac{1}{n}\sum_{i=1}^{n} \mathbf{W}_{V_i}\mathbf{H}^{(0)}\boldsymbol{\alpha}_i[:, n-1] \\ &= \mathbf{H}^{(0)}[n-1] \underbrace{-\mathbf{H}^{(0)}[n-1]}_{k\text{-th head}} + \underbrace{\sum_{i=0}^{k-1} \mathbf{W}_{V_i}\mathbf{H}^{(0)}\boldsymbol{\alpha}_i[:, n-1]}_{0,\dots,k-1\text{-head}}. \\ &= \begin{pmatrix} \mathbf{x}_0 & \cdots & \mathbf{x}_{k-1} & \mathbf{0} \end{pmatrix}^\top \end{aligned} \tag{16}$$

Since the tokens in the vocabulary $\mathcal{X}$ are non-parallel vectors, their concatenation should also be non-parallel for different tokens. Therefore, we have successfully constructed a single-layer transformer with n heads that can map sequences to non-parallel vectors by concatenating the tokens based on their positional information. □

Based on the aforementioned derivation, we can construct our final single-layer transformer as follows: first, employ the attention weights from Lemma E.2 to reshape sequences into no-parallel vectors , and then employ the methodology from Lemma E.1 to construct a classifier layer that maps each vector to a distinct label.

F. Proofs for Section 5.2

In this section, we will prove the main results in Section 5.2, which shows that the transformer requires at least two layers to perform successful reasoning. We will first prove that the reasoning task can never be perfectly resolved using single-layer transformer, no matter how many heads are included. Then, we show that two-layer transformers are capable of performing the designed reasoning tasks with perfect accuracy, by proving the existence of a set of attention weight matrices.

F.1. Proofs of Theorem 5.2

Proof. In order to prove the inability of the single-layer transformer, we will design a specific task and show that any single-layer transformer cannot achieve perfect test accuracy. In particular, We consider a task that contains 2 question $\mathbf{a}, \mathbf{b}$ and 2 answer $\mathbf{x}, \mathbf{y}$, take the response sign as "=". Then, the dataset $\mathcal{D}_{\texttt{toy-icl}} = \{\mathbf{E}^{(0)}, \mathbf{E}^{(1)}, \mathbf{E}^{(2)}, \mathbf{E}^{(3)}\}$ can be denote as follows:

1. input : $\mathbf{a} = \mathbf{x}\ \mathbf{b} = \mathbf{y}\ \mathbf{a} =$ target : $\mathbf{x}$
2. input : $\mathbf{a} = \mathbf{y}\ \mathbf{b} = \mathbf{x}\ \mathbf{b} =$ target : $\mathbf{x}$
3. input : $\mathbf{a} = \mathbf{y}\ \mathbf{b} = \mathbf{x}\ \mathbf{a} =$ target : $\mathbf{y}$
4. input : $\mathbf{a} = \mathbf{x}\ \mathbf{b} = \mathbf{y}\ \mathbf{b} =$ target : $\mathbf{y}$

It can be shown that these 4 sequences are **dependent**, as we defined in Definition D.2. To verify this, we first have that all these sequences end with the same token `"="`, then, it holds that

$$\mathbf{E}^{(0)} \oplus \mathbf{E}^{(1)} = ([\mathbf{a}, \mathbf{a}], [\texttt{"="}, \texttt{"="}], [\mathbf{x}, \mathbf{y}], [\mathbf{b}, \mathbf{b}], [\texttt{"="}, \texttt{"="}], [\mathbf{x}, \mathbf{y}], [\mathbf{a}, \mathbf{b}]) = \mathbf{E}^{(2)} \oplus \mathbf{E}^{(3)}.$$

Then, for any single layer transformer `TF`, we can leverage Proposition D.3 and then get that the output representations for the last token corresponding to all sequences are linearly dependent, i.e.,

$$\mathbf{o}^{(0)} + \mathbf{o}^{(1)} = \mathbf{o}^{(2)} + \mathbf{o}^{(3)}.$$

Then we are ready to show that there doesn't exist a single-layer transformer that can reason all these 4 examples. We will prove this by contradiction. First , suppose that there exists such a transformer `TF` that can correctly reason all these examples, then the output of the transformer will have the maximum output corresponding to the desired answer. Let i_x and i_y be the indices of the transformer output corresponding to the target $\mathbf{x}$ and $\mathbf{y}$ respectively, it shall hold that

$$\mathbf{o}^{(0)}[i_x] > \mathbf{o}^{(0)}[i_y], \quad \mathbf{o}^{(1)}[i_x] > \mathbf{o}^{(1)}[i_y], \quad \mathbf{o}^{(2)}[i_y] > \mathbf{o}^{(2)}[i_x], \quad \mathbf{o}^{(3)}[i_y] > \mathbf{o}^{(3)}[i_x]. \tag{17}$$

Besides, by linear dependency, we have

$$\mathbf{o}^{(0)} - \mathbf{o}^{(2)} = \mathbf{o}^{(3)} - \mathbf{o}^{(1)}. \tag{18}$$

Combining Equation (17) and Equation (18), we have the following contradiction:

$$(\mathbf{o}^{(0)} - \mathbf{o}^{(2)})[i_x] > (\mathbf{o}^{(0)} - \mathbf{o}^{(2)})[i_y] > (\mathbf{o}^{(3)} - \mathbf{o}^{(1)})[i_y] > (\mathbf{o}^{(3)} - \mathbf{o}^{(1)})[i_x] > (\mathbf{o}^{(0)} - \mathbf{o}^{(2)})[i_x]. \tag{19}$$

Therefore, this implies that no single-layer transformer can correctly reason all of these four sequences. In other words, if these four sequences appear with equal probability, the reasoning accuracy achieved by any single-layer transformer will be upper bounded by $3/4$.

□

F.2. Proof for Theorem 5.3

In this section we will show that a 2-layer transformer can perfectly perform the reasoning tasks, when provided with proper weights. In particular, we will construct such a transformer by following a copy-matching process, that is, the first layer of transformer copies the answer to the corresponding question ahead of them, and then the second layer searches these question-answer pairs and chooses the one with the highest similarity, i.e. having the same token, then the classifier layer projects the representation embedding to the answer.

Proof. Recall the data construction process, we consider an input sequence with k question-answer pairs, which is denoted as $\mathbf{H}^{(0)} \in \mathbb{R}^{d' \times n}$ ($n = 3k + 2$):

$$\mathbf{H}^{(0)} = \begin{pmatrix} \overbrace{\mathbf{q}_0 & \mathbf{0} & \mathbf{0}}^{Q-R-A} & \cdots & \mathbf{q}_{k-1} & \mathbf{0} & \mathbf{0} & \mathbf{q}_r & \mathbf{0} \\ \mathbf{0} & 1 & \mathbf{0} & \cdots & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{a}_0 & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{a}_{k-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_0 & \mathbf{p}_1 & \mathbf{p}_2 & \cdots & \mathbf{p}_{3k-3} & \mathbf{p}_{3k-2} & \mathbf{p}_{3k-1} & \mathbf{p}_{3k} & \mathbf{p}_{3k+1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{matrix} \rightarrow d_1 \times n \\ \rightarrow 1 \times n \\ \rightarrow d_2 \times n \\ \rightarrow n \times n \\ \rightarrow (d' - n - d) \times n \end{matrix},$$

where $[\mathbf{q}_k; \mathbf{0}]$ and $[\mathbf{0}; \mathbf{a}_k]$ denote the embeddings for the k-th question token and k-th answer token respectively, $[\mathbf{0}; 1; \mathbf{0}]$ denotes the embedding for the response sign, $r \in [k]$ is a random choose question index. Without loss of generality, we will set $r = 0$ in the following proof to illustrate how our construction works:

$$\mathbf{H}^{(0)} = \begin{pmatrix} \overbrace{\mathbf{q}_0 \quad \mathbf{0} \quad \mathbf{0}}^{Q-R-A} & \cdots & \mathbf{q}_0 & \mathbf{0} \\ \mathbf{0} \quad 1 \quad \mathbf{0} & \cdots & \mathbf{0} & 1 \\ \mathbf{0} \quad \mathbf{0} \quad \mathbf{a}_0 & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_0 \quad \mathbf{p}_1 \quad \mathbf{p}_2 & \cdots & \mathbf{p}_{3k} & \mathbf{p}_{3k+1} \\ \mathbf{0} \quad \mathbf{0} \quad \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix} \xrightarrow[\text{Q-A block}]{\texttt{TF}_1 \text{ copy among}} \begin{pmatrix} \overbrace{\mathbf{q}_0}^{Q-R-A} & \mathbf{q}_0 & \mathbf{q}_0 & \cdots & \mathbf{q}_0 & \overbrace{\mathbf{q}_0}^{Q-R} \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ \mathbf{a}_0 & \mathbf{a}_0 & \mathbf{a}_0 & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \end{pmatrix} \xrightarrow[\text{match col with same Q}]{\texttt{TF}_2 \text{ last token}} \begin{pmatrix} \vdots \\ \mathbf{a}_0 \\ \vdots \end{pmatrix}$$

First we construct an instructive attention matrix $\boldsymbol{\alpha}^{(1)} \in \mathbb{R}^{n\times n}$ by set $\mathbf{W}_{QK}^{(1)}$ follow the method in Lemma C.1:

$$\boldsymbol{\alpha}^{(1)} = \begin{pmatrix} \mathbf{A}_{3\times 3} & & 0 \\ & \ddots_{k \text{ times}} & \\ 0 & & \mathbf{A}_{2\times 2} \end{pmatrix} \quad \mathbf{A}_{3\times 3} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \quad \mathbf{A}_{2\times 2} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

And we set $\mathbf{W}_V^{(1)} = \mathbf{I}$, then the first layer only performs the copying operation based on the positions of tokens, and the output of the first layer $\mathbf{H}^{(1)}$ becomes

$$\mathbf{H}^{(1)} = \mathbf{H}^{(0)} + \mathbf{H}^{(0)}\boldsymbol{\alpha}^{(1)} = \begin{pmatrix} \mathbf{q}_0 & \mathbf{q}_0 & \mathbf{q}_0 & \cdots & \mathbf{q}_{k-1} & \mathbf{q}_0 & \mathbf{q}_0 \\ 1 & 1 & 1 & \cdots & 1 & 1 & 1 \\ \mathbf{a}_0 & \mathbf{a}_0 & \mathbf{a}_0 & \cdots & \mathbf{a}_{k-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_{0,1,2} & \mathbf{p}_{0,1,2} & \mathbf{p}_{0,1,2} & \cdots & \mathbf{p}_{3k-3,3k-2,3k-1} & \mathbf{p}_{3k,3k+1} & \mathbf{p}_{3k,3k+1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix},$$

where $\mathbf{p}_{i,j,\ldots} = \mathbf{p}_i + \mathbf{p}_i + \ldots$ means the sum of several positional encoding. In this way we copy the question, response sign and answer into a vector, making it possible for the match and carry step for the next layer.

Because our question and answer are one-hot vector, so $\mathbf{q}_i^\top \mathbf{q}_j = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases}$, so we can choose $\mathbf{W}_{QK}^{(2)} \in \mathbb{R}^{d'\times d'}$ as

$$\mathbf{W}_{QK}^{(2)} = \begin{pmatrix} \mathbf{I}_{d_1\times d_1} & \mathbf{0}_{d_1\times(d'-d_1)} \\ \mathbf{0}_{(d'-d_1)\times d_1} & \mathbf{0}_{(d'-d_1)\times(d'-d_1)} \end{pmatrix},$$

and set $\mathbf{W}_V^{(2)} = \mathbf{I}$, then the last output for the 2-second layer $\mathbf{H}^{(2)}[n-1]$

$$\mathbf{H}^{(2)}[n-1] = \mathbf{H}^{(1)}[n-1] + \mathbf{H}^{(1)}\sigma\left((\mathbf{H}^{(1)})^\top \mathbf{W}_{QK}^{(2)} \mathbf{H}^{(1)}[n-1]\right) = [6\mathbf{q}_0; 6; 3\mathbf{a}_0; 3\mathbf{p}_{3k,3k+1} + 3\mathbf{p}_{0,1,2}],$$

we then set the classification $\mathbf{W}_O = [\mathbf{0}_{d_2\times(d_1+1)}, \mathbf{I}_{d_2}, \mathbf{0}_{d_2\times(n-(d_1+d_2+1))}]^\top$, in this way, the prediction result of our construct transformer $\texttt{TF}$ is

$$\texttt{TF}(\mathbf{H}^{(0)}) = \mathbf{a}_1.$$

□

G. Proofs for Section 5.3

G.1. Proofs for Theorem 5.4

In this section, we delve into the reasons why transformers struggle to generalize on different templates, similar to their limitations in reasoning, the primary constraint on the transformer's ability arises from the linear dependence of the predicted results, as mentioned in Proposition D.3. Let's recall that a transformer is considered capable of generating output for a specific template if it can accurately classify **all** sequences generated by that template, so we achieve this by aggregating the prediction results and demonstrating that their sum must be linearly dependent, leading the failure of single-layer of transformer in generalization.

Lemma G.1. *For any template* $\mathbf{t}$ *and* ***all possible*** *sequences set generated by any :* $\mathcal{D}^{(\mathbf{t})}_{\texttt{tmpl}} = \{\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_{n^{(\mathbf{t})}-1}\}$*, each real world token* $\mathbf{x} \in \mathcal{X}$ *occur in* $\mathcal{D}^{(\mathbf{t})}_{\texttt{tmpl}}$ *at each position* $\frac{n^{(\mathbf{t})}}{|\mathcal{X}|}$ *times.*

Proof. The observation in Lemma G.1 is straightforward, as the sequence generated by the same template is token-symmetric: for any sequence $(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1})$ and any permutation $\pi : \mathcal{X} \to \mathcal{X}$, $(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1})$ and $(\pi(\mathbf{x}_0), \pi(\mathbf{x}_1), \ldots, \pi(\mathbf{x}_{n-1}))$ belong to the same template. Therefore, for each position, each token should occur in the $\mathcal{D}^{(\mathbf{t})}_{\texttt{tmpl}}$ with the same frequency, resulting in $\frac{n^{(\mathbf{t})}}{|\mathcal{X}|}$ occurrences for each token at any position. □

Example: Let $\mathcal{X} = \{a, b, c\}$ and two templates of length 2 $\{\alpha\alpha, \alpha\beta\}$. Then in total there are 3 sequences generated by $\alpha\alpha$: $\{aa, bb, cc\}$, each token occurs $\frac{n^{(\mathbf{t})}}{|\mathcal{X}|} = \frac{3}{3} = 1$ time at each position. For template $\alpha\beta$, there are 6 sequences $\{ab, ac, ba, bc, ca, cb\}$, each token occurs $\frac{6}{3} = 2$ times at each position.

Lemma G.2. *Given any two different templates* $\mathbf{t}, \mathbf{t}'$*, and* ***all possible*** *sequences generated* $\mathcal{D}^{(\mathbf{t})}_{\texttt{tmpl}} = \{\mathbf{X}_0, \ldots, \mathbf{X}_{n^{(\mathbf{t})}-1}\}$*,* $\mathcal{D}^{(\mathbf{t}')}_{\texttt{tmpl}} = \{\mathbf{X}'_0, \ldots, \mathbf{X}'_{n^{(\mathbf{t}')}-1}\}$*, there exist* $\lambda, \lambda' \in \mathbb{Z}_+$ *such that the combined occurrence of each token at each position in* $\mathcal{D}^{(\mathbf{t})}_{\texttt{tmpl}}$ *is equal to the combined occurrence of each token at each position in* $\mathcal{D}^{(\mathbf{t}')}_{\texttt{tmpl}}$*:*

$$\sqcap_{i\in[n^{(\mathbf{t})}]}(\lambda \otimes \mathbf{X}_i) = \sqcap_{j\in[n^{(\mathbf{t}')}]}(\lambda' \otimes \mathbf{X}_j), \tag{20}$$

where $\sqcap$ *denotes the token combination operation defined in Definition D.2, which computes the occurrence of each token at each position.*

Proof. Based on the observation in Lemma G.1, Lemma G.2 is straightforward. Since each token occurs $\frac{n^{(\mathbf{t})}}{|\mathcal{X}|}$ times at each position, we can let $\lambda = n^{(\mathbf{t}')}$ and $\lambda' = n^{(\mathbf{t})}$. Then, for the left part $\sqcap_{i\in[n^{(\mathbf{t})}]}(\lambda \otimes \mathbf{X}_i)$, each token occurs $n^{(\mathbf{t}')}\frac{n^{(\mathbf{t})}}{|\mathcal{X}|}$ times at each position. Similarly, for the right part $\sqcap_{j\in[n^{(\mathbf{t}')}]}(\lambda \otimes \mathbf{X}_j)$, each token also occurs $n^{(\mathbf{t})}\frac{n^{(\mathbf{t}')}}{|\mathcal{X}|} = n^{(\mathbf{t}')}\frac{n^{(\mathbf{t})}}{|\mathcal{X}|}$ times. This completes the proof. □

Example: Let $\mathcal{X} = \{a, b, c\}$ and two templates of length 2 $\{\alpha\alpha, \alpha\beta\}$. Then in total there are $n^{\alpha\alpha} = 3$ sequences generated by $\alpha\alpha$: $\{aa, bb, cc\}$, each token occurs 1 time at each position. For template $\alpha\beta$, there are $n^{\alpha\beta} = 6$ sequences $\{ab, ac, ba, bc, ca, cb\}$, each token occurs 2 times at each position. We can find positive integers $\lambda_1 = 6$ and $\lambda_2 = 3$ such that $3(ab \oplus ac \oplus ba \oplus bc \oplus ca \oplus cb) = 6([a, b, c], [a, b, c], [a, b, c]) = 6(aa \oplus bb \oplus cc)$. Note that in Equation 20, both sides have $n^{(\mathbf{t})}n^{(\mathbf{t}')}$ tokens. If we add a response sign at the end of each sequence, i.e., $\{aa, bb, cc\} \xrightarrow{\text{add sign } r} \{aar, bbr, ccr\}$, Lemma G.2 still holds.

Since the last token of the sequence input into the transformer is the same response sign, based on Lemma G.2 and Definition D.2, for any two templates, the sequences generated by them are dependent. This dependence leads to the following lemma:

Lemma G.3. *Given any two different templates* $\mathbf{t}, \mathbf{t}'$*, and* ***all possible*** *sequences generated* $\mathcal{D}^{(\mathbf{t})}_{\texttt{tmpl}} = \{\mathbf{X}_0, \ldots, \mathbf{X}_{n^{(\mathbf{t})}-1}\}$*,* $\mathcal{D}^{(\mathbf{t}')}_{\texttt{tmpl}} = \{\mathbf{X}'_0, \ldots, \mathbf{X}'_{n^{(\mathbf{t}')}-1}\}$*. For any single-layer transformer model* $\texttt{TF}$*, let* $\mathbf{o}^{(i)} = \texttt{TF}(\mathbf{X}_i)$ *and* $\mathbf{o}'^{(i)} = \texttt{TF}(\mathbf{X}'_i)$ *denote the model predictions for these two templates. Then we have:*

$$\lambda_1 \sum_{i=0}^{n^{(\mathbf{t}')}-1} \mathbf{p}^{(i)} = \lambda_2 \sum_{i=0}^{n^{(\mathbf{t}')}-1} \mathbf{p}'^{(i)}. \tag{21}$$

Proof. As the sequences generated by any two templates are dependent, we can apply Proposition D.3 to directly complete the proof. □

Note that any sequence generated by the same template should be classified by the same template. Therefore, if the model can generalize on both templates, the sum of the prediction output for template $\mathbf{t}$, $\sum_{i=0}^{n^{(\mathbf{t})}-1} \mathbf{p}^{(i)}$, should have the maximum value at position y_1, and $\sum_{i=0}^{n^{(\mathbf{t}')}-1} \mathbf{p}'^{(i)}$ should have the maximum value at position y_2, where $y_1 \neq y_2$ since different templates belong to different classes. This contrasts with Lemma G.3, and thus, we can conclude that transformers cannot generalize on any two different templates.

G.2. Proof for Theorem 5.5

In this section, we will construct 2-layer transformer that can generalize on the template task, our construct transformer can first parse the sequence into the template, and then use the memorization ability of one-layer transformer, mapping the template to the corresponding label, follow such parsing-mapping procedure, our constructed transformer can generalize on the template task.

Recall the data construction process, we consider an input sequence generated by a template length k, a $d' \times n$ matrix $\mathbf{H}^{(0)}$ where $n = k+1$:

$$\mathbf{H}^{(0)} = \begin{pmatrix} \overbrace{\mathbf{x}_0 & \mathbf{x}_1 & \cdots & \mathbf{x}_{k-1}}^{\text{sequence by template}} & \overbrace{\mathbf{0}}^{\text{response sign}} \\ 0 & 0 & \cdots & 0 & 1 \\ \mathbf{p}_0 & \mathbf{p}_1 & \cdots & \mathbf{p}_{k-1} & \mathbf{p}_k \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{matrix} \to d \times n \\ \to 1 \times n \\ \to n \times n \\ \to (d' - (n+1+d)) \times n \end{matrix}$$

We illustrate our construction by using a template length 3 $\alpha\beta\beta$,which generates the sequence $(\mathbf{a}, \mathbf{b}, \mathbf{b})$:

$$\mathbf{H}^{(0)} = \begin{pmatrix} \overbrace{\mathbf{a} & \mathbf{b} & \mathbf{b}}^{\text{from template } \alpha\beta\beta} & \overbrace{\mathbf{0}}^{\text{response sign}} \\ 0 & 0 & 0 & 1 \\ \mathbf{p}_0 & \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \xrightarrow[\text{check position with the same token}]{\texttt{TF}_1} \begin{pmatrix} \mathbf{a} & \mathbf{b} & \mathbf{b} & \mathbf{0} \\ 0 & 0 & 0 & 1 \\ \mathbf{p}_0 + \mathbf{p}_0 & \mathbf{p}_1 + \mathbf{p}_{1,2} & \mathbf{p}_2 + \mathbf{p}_{1,2} & \mathbf{p}_3 + \mathbf{p}_3 \\ \underbrace{\mathbf{0} & \mathbf{0} & \mathbf{0}}_{\text{positional encoding as template representation}} & \mathbf{0} \end{pmatrix}$$

$$\xrightarrow[\text{map template to vectors}]{\texttt{TF}_2} \begin{pmatrix} \vdots \\ (200)_{(3)} \\ (021)_{(3)} \\ (012)_{(3)} \\ \vdots \end{pmatrix} \text{encode each row to a ternary number}$$

First we set our transformer use attention mechanism parse each sequence into template, recall the match layer in Appendix F.2, each token only focus on the position of the same token, in this way, the position dimension can be considered as a dimension of template sequence:

$$\mathbf{W}_{QK}^{(1)} = \begin{pmatrix} \mathbf{I}_{d\times d} & \mathbf{0}_{d\times(d'-d)} \\ \mathbf{0}_{(d'-d)\times d} & \mathbf{0}_{(d'-d)\times(d'-d)} \end{pmatrix},$$

$$\mathbf{W}_V^{(1)} = \begin{pmatrix} -\mathbf{I}_{(d+1)\times(d+1)} & \mathbf{0}_{(d+1)\times(d'-(d+1))} \\ \mathbf{0}_{(d'-(d+1))\times(d+1)} & \mathbf{0}_{(d'-(d+1))\times(d'-(d+1))} \end{pmatrix} + \begin{pmatrix} \mathbf{0}_{(d+1)\times(n)} & \mathbf{0}_{(d+1)\times(d'-n)} \\ \mathbf{I}_{n\times n} & \mathbf{0}_{n\times(d'-n)} \\ \mathbf{0}_{(d'-(n+1+d))\times(n)} & \mathbf{0}_{(d'-(n+1+d))\times(d'-n)} \end{pmatrix}.$$

Then the resulting sequence to be input to the second layer is

$$\mathbf{H}^{(1)} = \mathbf{H}^{(0)} + \mathbf{W}_V^{(1)}\mathbf{H}^{(0)}\boldsymbol{\alpha}^{(1)} = \mathbf{H}^{(0)} + \begin{pmatrix} -\mathbf{x}_0 & -\mathbf{x}_1 & \cdots & -\mathbf{x}_{k-1} & \mathbf{0} \\ 0 & 0 & \cdots & 0 & -1 \\ \mathbf{p}'_0 & \mathbf{p}'_1 & \cdots & \mathbf{p}'_{k-1} & \mathbf{p}_k \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix}$$

$$= \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ 0 & 0 & \cdots & 0 & 0 \\ \mathbf{p}'_0 + \mathbf{p}_0 & \mathbf{p}'_1 + \mathbf{p}_1 & \cdots & \mathbf{p}'_{k-1} + \mathbf{p}_{k-1} & \mathbf{p}_k + \mathbf{p}_k \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix}.$$

The second part can be seen as a sequence of template, take $k = 4$ and template be $\alpha\beta\alpha\gamma$ as example, then $\mathbf{p}'_0 = \mathbf{p}'_2 = \mathbf{p}_0 + \mathbf{p}_2, \mathbf{p}'_1 = \mathbf{p}_1, \mathbf{p}'_3 = \mathbf{p}_3$, then the result template representation is $\begin{pmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$. Note that $\mathbf{p}'_i = \mathbf{p}'_j$ if and only if $\mathbf{x}_i = \mathbf{x}_j$. We can further prove that the output sequence parsed by the first layer of the transformer satisfies the following:

- Sequence from the same template have the same representation.
- Different templates have different, no-parallel representation matrix.

The first point is straightforward since this representation is based on positional encoding and is token-invariant. For the second point, since the elements in this representation only consist of the values $0, 1, 2$, and the diagonal value is always 2 for any two representations $\mathbf{H}^{(1)}$ and $\mathbf{H}^{'(1)}$, if there exists a λ such that $\mathbf{H}^{(1)} = \lambda \mathbf{H}^{'(1)}$, then we must have $\lambda = 1$ and $\mathbf{H}^{(1)} = \mathbf{H}^{'(1)}$. Therefore, each representation is non-parallel.

The role of second layer, incorporate with the classification layer, is to map the sequence to its corresponding label, as we define different template have different label, and based on the two property of the output of the first layer, we can utilize the memorization ability of the first layer transformer, as demonstrated in Appendix E, and following the construction procedure, construct a transformer with n attention heads that can correctly map each sequence to corresponding label.In addition to that, since template encoding is a more structured form of data compared to randomly assembled sequences, we provide one possible weight that can map the templates to non-parallel vectors using only a single attention head:

First we construct a inductive head with $\boldsymbol{\alpha} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ 3^{k-1} & 3^{k-2} & \cdots & 3^0 & 0 \end{pmatrix}^\top$, and set $\mathbf{W}_V^{(2)} = \mathbf{I}$, in this way the last token of the sencond layer is $\sum_{i=1}^{k} 3^{k-i}(\mathbf{p}_i + \mathbf{p}'_i)$, as each element of $\mathbf{p}_i + \mathbf{p}'_i$ only consists of three integers $0, 1, 2$, and $(\mathbf{p}_i + \mathbf{p}'_i)[i] = 2$, in this way, for each row of $\mathbf{H}^{(1)}$, we can encode it into a unique ternary number, for example, given a vector $[2, 1, 1, 0, 0]$, we can encode it to $2 \times 3^4 + 1 \times 3^3 + 1 \times 3^2 + 0 \times 3^1 + 0 \times 3^0 = 21100_{(3)}$, and the representation for $\alpha\beta\alpha\gamma$, we encode it row by row: $\begin{pmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} (2010)_{(3)} \\ (0200)_{(3)} \\ (1020)_{(3)} \\ (0002)_{(3)} \end{pmatrix}$. Now consider the representation at position k, as $(\mathbf{p}_k + \mathbf{p}'_k)[k-1] = 2$ for all representation, the resulting vector at position k should be $(\ldots 2)_{(3)}$, in this way we can ensure different template will project into no-parallel vectors, as if two template representaion vector is parallel, they must have the **same** representation vector, which further indicate they must have the same template representation at each row.

Then utilizing Lemma E.1, we can construct a $\mathbf{W}_C \in \mathbb{R}^{d' \times C}$ map each vector to their distinct label. In this way, a part from the n attention head transformer, we further proof there exists a 2-layer transformer with single head that can accomplish our generalization task.

H. Construction for Section 5.4

In this section, we provide a construction for a 3-layer transformer that is capable of handling contextual generalization task. Our constructed transformer follows a parsing-copy-match process, the transformer parses each question into corresponding template, this is achieved by utilizing the parsing process in the template generalization task, we use the constrained attention to force the model focuses only on tokens that belong to the same question block. Next, in the second layer, we mix the question and corresponding answer together by utilizing inductive attention. Then the final layer retrieve the corresponding question-answer representation and transform it into the final answer.

Let us consider an input sequence with k question-answer pairs, each question being of sequence length l. We represent this input sequence as $\mathbf{H}^{(0)} \in \mathbb{R}^{d' \times n}$, where $n = (l+2)k + l + 1$.

$$\mathbf{H}^{(0)} = \begin{pmatrix} \overbrace{\mathbf{x}_0^{(0)} & \cdots & \mathbf{x}_{l-1}^{(0)}}^{\text{Question}} & \overbrace{\mathbf{0}}^{\text{Response Sign}} & \overbrace{\mathbf{0}}^{\text{Answer}} & \cdots & \mathbf{x}_0^{\prime(r)} & \cdots & \mathbf{x}_{l-1}^{\prime(r)} & \mathbf{0} \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & \cdots & 0 & 1 \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{a}_0 & \cdots & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_0 & \cdots & \mathbf{p}_{l-1} & \mathbf{p}_l & \mathbf{p}_{l+1} & \cdots & \mathbf{p}_{(l+2)k} & \cdots & \mathbf{p}_{(l+2)k+l-1} & \mathbf{p}_{(l+2)k+l} \\ \underbrace{\mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots}_{(Q-R-A)\times k} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{matrix} \to d_1 \times n \\ \to 1 \times n \\ \to d_2 \times n \\ \to n \times n \\ \to (d' - n - d) \times n \end{matrix}$$

Where $r \in [k]$ is a random choose question index, we can set $r = 1$, $k = 2$ and $l = 2$, which means there are only two templates $\alpha\alpha$ and $\alpha\beta$, to illustrate how our construction works:

$$\mathbf{H}^{(0)} = \begin{pmatrix} \overbrace{\mathbf{a} & \mathbf{a}}^{\text{Question}} & \overbrace{\mathbf{0}}^{\text{Response Sign}} & \overbrace{\mathbf{0}}^{\text{Answer}} & \mathbf{a} & \mathbf{b} & \mathbf{0} & \mathbf{0} & \mathbf{b} & \mathbf{b} & \mathbf{0} \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{a}_0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{a}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_0 & \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 & \mathbf{p}_4 & \mathbf{p}_5 & \mathbf{p}_6 & \mathbf{p}_7 & \mathbf{p}_8 & \mathbf{p}_9 & \mathbf{p}_{10} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}$$

$$\xrightarrow[\substack{\text{check token} \\ \text{among Q-A block \&} \\ \text{align representation}}]{\texttt{TF}_1} \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{a}_0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{a}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 2 & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 1 & 0 \\ 1 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 2 & 0 \\ \underbrace{\vdots & \vdots}_{\substack{\text{template} \\ \alpha\alpha}} & \vdots & \vdots & \underbrace{\vdots & \vdots}_{\substack{\text{template} \\ \alpha\beta}} & \vdots & \vdots & \underbrace{\vdots & \vdots}_{\substack{\text{template} \\ \alpha\alpha}} & \vdots \end{pmatrix}$$

$$\xrightarrow[\substack{\text{copy answer} \\ \text{compare each template}}]{\texttt{TF}_2} \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{a}_0 & \mathbf{a}_0 & \mathbf{0} & \mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a}_1 & \mathbf{0} & \mathbf{a}_1 & \mathbf{0} & \mathbf{0} & \mathbf{a}_0 + \mathbf{a}_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 1 & 2 & 1 \\ \underbrace{\vdots & \vdots}_{\substack{\text{same template} \\ \text{template row all zero}}} & \vdots & \vdots & \underbrace{\vdots & \vdots}_{\substack{\text{different template} \\ \text{exist no-zero elements}}} & \vdots & \vdots & \underbrace{\vdots & \vdots}_{\substack{\text{template} \\ \alpha\alpha}} & \vdots \end{pmatrix}$$

$$\xrightarrow[\substack{\text{focus on col that template} \\ \text{exists no-zero elements}}]{\texttt{TF}_3\text{, last token}} \begin{pmatrix} \vdots \\ \mathbf{a}_0 + \mathbf{a}_1 - \gamma \mathbf{a}_1 \\ \vdots \end{pmatrix} (\gamma \geq 1)$$

In the first layer, we parse each question into a template. To achieve this, we set the weight $\mathbf{W}_{QK}^{(1)}$ as follows:

$$\mathbf{W}_{QK}^{(1)} = \begin{pmatrix} \mathbf{I}_{d_1 \times d_1} & \mathbf{0}_{d_1 \times (d'-d_1)} \\ \mathbf{I}_{(d'-d_1) \times d_1} & \mathbf{0}_{(d'-d_1) \times (d'-d_1)} \end{pmatrix} + \begin{pmatrix} \mathbf{0}_{d \times d} & \mathbf{0}_{d \times n} & \mathbf{0}_{d \times (d'-(n+d))} \\ \mathbf{0}_{n \times d} & \mathbf{W}_{pp} & \mathbf{0}_{n \times (d'-(n+d))} \\ \mathbf{0}_{(d'-(n+d)) \times d} & \mathbf{0}_{(d'-(n+d)) \times n} & \mathbf{0}_{(d'-(n+d)) \times (d'-(n+d))} \end{pmatrix}.$$

Here, $d = d_1 + d_2 + 1$, and $\mathbf{W}_{pp}$ is an $n \times n$ matrix. As discussed in Lemma C.1, we can set:

$$\mathbf{W}_{pp} = \begin{pmatrix} \mathbf{0}_{(l+2)\times(l+2)} & & -\infty \\ & \ddots_{k \text{ times}} & \\ -\infty & & \mathbf{0}_{(l+1)\times(l+1)} \end{pmatrix}.$$

This way, the attention is constrained to each question-answer block, making it easier to parse each question into a template. We then set

$$\mathbf{W}_V^{(1)} = \begin{pmatrix} \mathbf{0}_{(d'-l-2)\times d} & \mathbf{0}_{(d'-l-2)\times(l+2)} & \cdots & \mathbf{0}_{(d'-l-2)\times(l+1)} \\ \mathbf{0}_{(l+2)\times d} & \underbrace{\mathbf{I}_{(l+2)\times(l+2)} \quad \cdots}_{k \text{ times}} & & \mathbf{I}_{(l+2)\times(l+1)} \end{pmatrix}.$$

Here, we assume that the hidden size d' is large enough, so that $d' - (d+n) > l+2$. This enables the model to save the representation disentangled. As a result, the output of the first layer has the same positional encoding as the input sequence.

In the second layer, we perform a pre-match and copy procedure by constructing an instructive attention $\boldsymbol{\alpha}^{(2)} \in \mathbb{R}^{n\times n}$.

$$\boldsymbol{\alpha}^{(2)} = \begin{pmatrix} \mathbf{A}_{(l+2)\times(l+2)} & & 0 \\ & \ddots_{(k-1) \text{ times}} & \\ 0 & & \mathbf{A}_{(l+1)\times(l+1)} \end{pmatrix}^\top + \left.\begin{pmatrix} \mathbf{0}_{(l+2)\times(d-(l+1))} & \mathbf{B}_{(l+2)\times(l+1)} \\ \vdots & \vdots \\ \mathbf{0}_{(l+1)\times(d-(l+1))} & \mathbf{0}_{(l+1)\times(l+1)} \end{pmatrix}\right\} k \text{ times}^\top.$$

Here, $\mathbf{A}_{(l+2)\times(l+2)} = \begin{pmatrix} \mathbf{0}_{(l+2)\times(l+1)} & \mathbb{1}_{(l+2)\times 1} \end{pmatrix}$, $\mathbf{A}_{(l+1)\times(l+1)} = \begin{pmatrix} \mathbf{0}_{l\times(l+1)} \\ \mathbb{1}_{1\times(l+1)} \end{pmatrix}$, and $\mathbf{B}_{(l+2)\times(l+1)} = \begin{pmatrix} \mathbf{I}_{(l+1)\times(l+1)} \\ \mathbf{0}_{1\times(l+1)} \end{pmatrix}$.

The left part of $\boldsymbol{\alpha}^{(2)}$ copies the answer to each column of the question (template), while the right part of the attention tries to compare each row of the template with the last template. As we set $\mathbf{W}_V^{(2)} = \begin{pmatrix} \mathbf{I}_{d\times d} & \mathbf{0}_{d\times(d'-d)} \\ \mathbf{0}_{(d'-d)\times d} & -\mathbf{I}_{(d'-d)\times(d'-d)} \end{pmatrix}$, if there is a template that is the same as the last one, then their template representation is zero. In this way, we perform a copy procedure, accompanied by the template pre-matching procedure in the second layer.

In the third layer, we utilize $2l$ attention heads. If any row of the template representation is not zero, there exist at least one attention head that subtracts the corresponding answer from the last token. To achieve this, we set the weight matrices as follows:

$$\mathbf{W}_{QK_i}^{(3)} = \begin{pmatrix} \mathbf{0}_{(d'-l)\times(d'-l)} & \mathbf{0}_{(d'-l)\times l} \\ \mathbf{0}_{l\times(d'-l)} & \mathbf{M}_i \end{pmatrix} \quad \mathbf{W}_{V_i}^{(3)} = -\mathbf{I}.$$

Here, $\mathbf{M}_i \in \mathbb{R}^{l\times l}$. For the first l heads ($0 \le i < l$), only the element at position (i,i) is 1 and the rest are zero. For $l \le i < 2l$, only the element at position $(i-l, i-l)$ is -1 and the rest are zero. Together with $\mathbf{W}_{V_i}^{(3)} = -\mathbf{I}$, the third layer will focus on the answer that has non-zero elements and subtract the corresponding answer from the final representation.

The output for the last token is as follows:

$$\mathtt{TF}_3(\mathbf{H}^{(3)})[n-1] = \begin{pmatrix} \vdots \\ \sum_{i=0}^{k-1} \mathbf{a}_i - \sum_{i=0, i\neq r}^{k-1} \gamma_i \mathbf{a}_i \\ \vdots \end{pmatrix} (\gamma_i \ge 1).$$

As we assume the tokens are one-hot, we can set the classifier layer as

$$\mathbf{W}_O = \begin{pmatrix} \mathbf{0}_{(d_1+1)\times d_2} & \mathbf{I}_{d_2\times d_2} & \mathbf{0}_{(d'-d\times d_2)} \\ \mathbf{0}_{(d_1+1)\times(c-d_2)} & \mathbf{0}_{d_2\times(c-d_2)} & \mathbf{0}_{(d'-d\times(c-d_2))} \end{pmatrix}.$$

By letting $c = d_2$, the final prediction result is $\sum_{i=0}^{k-1} \mathbf{a}_i - \sum_{i=0, i\neq r}^{k-1} \gamma_i \mathbf{a}_i \, (\gamma_i \geq 1)$, which has the largest element corresponding to the final answer $\mathbf{a}_r$.

I. Expanding from ReLU to Softmax Attention

When considering a transformer with a single layer and a single head, the results for a single-layer ReLU attention transformer still hold.

In the case of softmax attention, which replaces the activation function in Equation 2 from ReLU ($\sigma(x) = \max\{0, x\}$) to softmax ($\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$), the input sequence $\mathbf{X} \in \mathbb{R}^{d' \times n}$. For a single attention head with parameters $\mathbf{W}_V$ and $\mathbf{W}_{QK}$, we define $\sigma'(\mathbf{x}_j, \mathbf{x}_k) = \exp(\mathbf{x}_j^\top \mathbf{W}_{QK} \mathbf{x}_k)$. Equation 7 can be rewritten for softmax attention as follows:

$$\texttt{TF}_{\text{soft}}(\mathbf{H})[k] = \mathbf{x}_{n-1} + \sum_{i=0}^{n-1} \frac{\mathbf{W}_V \mathbf{x}_i \sigma'(\mathbf{x}_i, \mathbf{x}_k)}{\sum_{j=0}^{n-1} \sigma'(\mathbf{x}_j, \mathbf{x}_k)} \tag{22}$$

Proposition I.1. *If the input sequences $\mathbf{X}^{(0)}, \mathbf{X}^{(1)}, \dots, \mathbf{X}^{(N-1)} \in \mathcal{X}^n$ are **dependent**, then for any single layer single head transformer with softmax attention $\texttt{TF}_{\textit{soft}}$, we have*

$$\lambda'_0(\texttt{TF}_{\textit{soft}}(\mathbf{X}^{(0)})[n-1]) + \lambda'_1(\texttt{TF}_{\textit{soft}}(\mathbf{X}^{(1)})[n-1]) + \cdots + \lambda'_{N-1}(\texttt{TF}_{\textit{soft}}(\mathbf{X}^{(N-1)})[n-1]) = \mathbf{0}.$$

where $\lambda'_i = \sum_{j=0}^{n-1} \sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1})\lambda_i$, $\{\lambda_i\}_{i=0}^{N-1}$ represents the coefficients defined in Definition D.2.

Proof. Consider the last token

$$\texttt{TF}_{\text{soft}}(\mathbf{X}^{(e)})[n-1] = \mathbf{x}_{n-1} + \sum_{i=0}^{n-1} \frac{\mathbf{W}_V \mathbf{x}_i^{(e)} \sigma'(\mathbf{x}_i^{(e)}, \mathbf{x}_{n-1})}{\sum_{j=0}^{n-1} \sigma'(\mathbf{x}_j^{(e)}, \mathbf{x}_{n-1})}$$

then we have

$$\begin{aligned} &\lambda'_0(\texttt{TF}_{\text{soft}}(\mathbf{X}^{(0)})[n-1]) + \lambda'_1(\texttt{TF}_{\text{soft}}(\mathbf{X}^{(1)})[n-1]) + \cdots + \lambda'_{N-1}(\texttt{TF}_{\text{soft}}(\mathbf{X}^{(N-1)})[n-1]) \\ =&\mathbf{x}_{n-1}(\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \lambda_i(\sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1}))) + (\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(\lambda_i \mathbf{W}_V \mathbf{x}_j^{(i)} \sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1})) \end{aligned} \tag{23}$$

Similar with the proof techique in Proposition D.3 we can divide the sequences into two groups $\mathcal{I}_+$ and $\mathcal{I}_-$ so in each position j, both side have the same occurrence for each token:

$$\sqcap_{i\in\mathcal{I}_+}(\lambda_i \otimes \mathbf{x}_j^{(i)}) = \sqcap_{i\in\mathcal{I}_-}(-\lambda_i \otimes \mathbf{x}_j^{(i)}) := \mathcal{S}_j, \tag{24}$$

here we use $\mathcal{S}_j$ to denote the tokens occurrences at position j, note that

$$\lambda \mathbf{W}_V \mathbf{x}_j^{(i)} \sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1}) = \sum_{\mathbf{s}\in(\lambda\otimes\mathbf{x}_j^{(i)})} \mathbf{W}_V \mathbf{s} \sigma'(\mathbf{s}, \mathbf{x}_{n-1})$$

$$\lambda \sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1}) = \sum_{\mathbf{s}\in(\lambda\otimes\mathbf{x}_j^{(i)})} \sigma'(\mathbf{s}, \mathbf{x}_{n-1})$$

, so we can derive the following equation:

$$\begin{aligned}
&\sum_{j=0}^{n-1} \lambda_i(\sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1})) \\
=&\sum_{i\in\mathcal{I}_+} \lambda_i\sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1}) - \sum_{i\in\mathcal{I}_-} -\lambda_i\sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1}), \\
=&\sum_{\mathbf{s}\in\mathcal{S}_j} \sigma'(\mathbf{s}, \mathbf{x}_{n-1}) - \sum_{\mathbf{s}\in\mathcal{S}_j} \sigma'(\mathbf{s}, \mathbf{x}_{n-1}) \\
=&\mathbf{0}
\end{aligned} \tag{25}$$

$$\begin{aligned}
&\sum_{j=0}^{n-1} \lambda_i(\mathbf{W}_V\mathbf{x}_j^{(i)}\sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1})) \\
=&\sum_{i\in\mathcal{I}_+} \lambda_i(\mathbf{W}_V\mathbf{x}_j^{(i)}\sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1})) - \sum_{i\in\mathcal{I}_-} -\lambda_i(\mathbf{W}_V\mathbf{x}_j^{(i)}\sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1})), \\
=&\sum_{\mathbf{s}\in\mathcal{S}_j} (\mathbf{W}_V\mathbf{s}\sigma'(\mathbf{s}, \mathbf{x}_{n-1})) - \sum_{\mathbf{s}\in\mathcal{S}_j} (\mathbf{W}_V\mathbf{s}\sigma'(\mathbf{s}, \mathbf{x}_{n-1})) \\
=&\mathbf{0}
\end{aligned} \tag{26}$$

□

Based on Proposition I.1, we observe that the single layer single head softmax transformer shares a similar dependent property with the ReLU attention only transformer:

Proposition I.2. *If the input sequences $\mathbf{X}^{(0)}, \mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N-1)} \in \mathcal{X}^n$ are **dependent**, then for any single layer single head softmax transformer $\texttt{TF}_{soft}$, their prediction result $\mathbf{o}^{(0)}, \ldots, \mathbf{o}^{(N-1)}$*

$$\lambda'_0\mathbf{o}^{(0)} + \lambda'_1\mathbf{o}^{(1)} + \cdots + \lambda'_{N-1}\mathbf{o}^{(N-1)} = \mathbf{0}.$$

where $\lambda'_i = \sum_{j=0}^{n-1} \sigma'(\mathbf{x}_j^{(i)}, \mathbf{x}_{n-1})\lambda_i$, $\{\lambda_i\}_{i=0}^{N-1}$ represents the coefficients defined in Definition D.2.

Expanding our results from Theorem 5.2,Theorem 5.4 to the softmax attention only transformer, we can conclude that a single-layer single-head attention only transformer is incapable of handling our reasoning and generalization tasks.