Journal of Machine Learning **ISSN**: 2790-2048(e), 2790-203X(p)

# Anchor Function: A Type of Benchmark Functions for Studying Language Models

Zhongwang Zhang[1], Zhiwei Wang[1], Junjie Yao[1], Zhangchen Zhou[1], Xiaolong Li[1], Weinan E[2], and Zhi-Qin John Xu*[1]

[1]Institute of Natural Sciences, School of Mathematical Sciences, Shanghai Jiao Tong University, Shanghai 200240, China.
[2]Beijing International Center for Mathematical Research (BICMR), Peking University, Beijing 100871, China.

**Abstract.** Understanding transformer-based language models is becoming increasingly crucial, particularly as they play pivotal roles in advancing towards artificial general intelligence. However, language model research faces significant challenges, especially for academic research groups with constrained resources. These challenges include complex data structures, unknown target functions, high computational costs and memory requirements, and a lack of interpretability in the inference process, etc.
Drawing a parallel to the use of simple models in scientific research, we propose the concept of an anchor function. This is a type of benchmark function designed for studying language models in learning tasks that follow an "anchor-key" pattern. By utilizing the concept of an anchor function, we can construct a series of functions to simulate various language tasks. The anchor function plays a role analogous to that of mice in diabetes research, particularly suitable for academic research.
We demonstrate the utility of the anchor function with an example, revealing two basic operations by attention structures in language models: shifting tokens and broadcasting one token from one position to many positions. These operations are also commonly observed in large language models. The anchor function framework, therefore, opens up a series of valuable and accessible research questions for further exploration, especially for theoretical study.

**Keywords:**
Language models,
Transformer,
Anchor function,
Machine learning theory.

**Article Info.:**
Volume: X
Number: X
Pages: 1 - 39
Date: /2026
doi.org/10.4208/jml.250723

**Article History:**
Received: 23/07/2025
Accepted: 20/12/2025

**Communicated by:**
Qianxiao Li

## 1 Introduction

Studying a natural phenomenon in scientific research often parallels the complexities encountered in language model research. As Feynman famously argued, "What I cannot create, I do not understand", and Hinton's paper, "To recognize shapes, first learn to generate images" [12], both perspectives underline a fundamental principle in scientific inquiry: to truly comprehend a complex phenomenon, one should possess the capability to reproduce it. For example, mice are a valuable model for diabetes research due to their genetic and physiological similarities to humans, allowing for relevant gene function studies. Their short lifespan enables quicker observation of the disease process, enhancing research efficiency. Additionally, mice are cost-effective and easy to breed, reducing research costs.

*Corresponding author. xuzhiqin@sjtu.edu.cn

Importantly, their genes can be easily modified using genetic engineering, facilitating the study of specific genes in diabetes development. These advantages make mice an ideal choice for diabetes research.

This scientific approach to animal models is applicable to the study of language models. By designing careful and focused experiments, researchers can uncover significant characteristics of language models. For example, studies like Lake and Baroni [16] demonstrate the potential of language models to learn systematic compositionality. [1,2,10,34] analyze the mechanism of transformers on various designed linear regression tasks. [5,24,27] utilize the item-label sequence and [16] designs a primitive composition dataset. [28] generates the sequence from a Boltzmann distribution to train the transformer model. Such targeted experimentation can effectively isolate and analyze specific capabilities of language models, contributing to a deeper understanding of their functionalities and limitations.

In language-related tasks, a common pattern is the identification of anchor words that indicate specific operations to be performed on other tokens within the sentence. For instance, in phrases like "summation of 1, 16, 17" and "product of 1, 16, 17", the anchor words "summation" and "product" respectively signal the mathematical operations to be applied to the key tokens 1, 16, and 17. To better comprehend how neural networks manage the "anchor-operation" pattern, this paper introduces a concept referred to as the "anchor function" for studying language models. A basic example of an anchor function is the identity learning function. In this scenario, a designated number, which acts as an anchor prompt, appears only once in each input sequence. The output of this function is the number immediately following this specific anchor. For example, if we take "3" as the anchor, then for the sequence $(43,33,13,3,20,89,44,24,56)$, the output would be 20. All other numbers in the sequence are inconsequential to the result. This concept mirrors a task that finds out the name in the input sequence, for example, the output of "He is from China, his name is Mike, he likes reading." is "Mike" and the anchor is "name is".

Furthermore, one could design a task involving two different anchors, each directing the network to execute a distinct operation. This kind of task is reflective of composite tasks in language models, where multiple prompts or cues within a sequence guide the model to perform various operations, demonstrating the model's ability to handle complex instructions. The anchor function qualifies as an effective type of benchmark function for studying language models, offering, but are not limited to, the following several advantages:

- The anchor function can simulate a range of language tasks, demonstrating its versatility in reproducing various linguistic scenarios.
- It is cost-effective in terms of computation and memory, making it a feasible option for academic research groups with limited resources.
- With a well-defined target function, the anchor function provides a clear and straightforward objective for analysis.
- It allows for a distinct separation between training data and test data, essential for accurate model evaluation.

- Experiments with the anchor function can be conducted under strict control of variables, ensuring precise and reliable outcomes.
- This function is instrumental in studying both the generalization within a single task and the transfer of learning from known tasks to new, unseen tasks.
- Fully-connected networks struggle to learn tasks with logical structures, such as identity learning tasks, in contrast to language models, which sheds light on the differing capabilities of various network architectures.

These attributes make the anchor function a robust and insightful tool for probing the complexities and capabilities of language models.

In this work, we will showcase a range of anchor functions that effectively simulate various language tasks. These functions serve as tools to investigate the operational mechanisms of transformer networks in executing these tasks. Specifically, in the context of the identity learning anchor function, we have identified that two fundamental operations, executed by the attention structure, are critical. These operations involve shifting and broadcasting tokens. Our findings also reveal that these attention processes are also present in large language models (LLMs) such as Llama. We further illustrate that these operations can be achieved by manipulating the orientation of weight vectors within a high-dimensional space. We also utilize frequency principle [26, 38–40] and condensation [21, 46] of existing deep learning theory to understand the training of transformer networks. Building on the anchor functions, we outline a series of intriguing and significant issues that can be further explored in transformer networks using the anchor function framework. This approach opens up new avenues for understanding and enhancing the capabilities of transformer-based models, especially for theoretical study.

## 2 Related works

[33] evaluate the contribution of individual attention heads in the encoder to the overall performance of the model and propose a method to prune the attention heads. [8] analyze the attention mechanism of the Bert model and find that its attention head shows some pattern like specific positional offsets or focuses on delimiter tokens. [15] analyze Bert's encoding capabilities and attention matrix, propose the idea that Bert is over-parameterized, and improve Bert's performance by disabling certain attention layers.

[14] study the empirical scaling law of language model performance on cross-entropy loss. [23] prove that self-attention is a natural structure for dealing with sequence-to-sequence problems. [13] examine the ability of the transformer model to approximate sequential relationships, establish a universal approximation theory in the hypothesis space of the transformer, and provide an explicit approximation rate estimate.

[35] investigate the working mechanism of in-context learning (ICL) through an information flow lens. They demonstrate a mechanism by which ICL aggregates label descriptions into label words at shallow layers and uses label words as the reference for the final output at deeper layers.

[30] introduce a memory-augmented neural network to solve a meta-learning task. In such a task, each data episode has the same meta structure, but a different detail rule. [16] introduce a similar meta-learning for compositionality (MLC) to verify that the transformer network can achieve the systematic generalization of compositionality. In MLC, each sequence in the training or test data contains a different grammar generated by the same meta-grammar. As the training converges, the standard encoder-decoder transformer can predict the output of test data entirely dependent on the grammar in the input sequence in the given test data.

[9] notice that in a two-layer transformer neural network, some heads in the second layer attention matrix can notice the similarity between the current token and the previous tokens in the input sentence, then, the network would output the next token of the previous similar token, thereby completing tasks like "$\ldots, [A], [B], \ldots, [A] \rightarrow [B]$". Such a head that pays attention to previous similar tokens is called an "induction head". [24] point out that induction heads may be the primary mechanism for the transformer model to achieve in-context learning.

[5] utilize the Omniglot dataset [17] to show that the data distribution can affect the trade-off between in-weight learning and in-context learning. [18] explore the ability of a single transformer layer to learn nearest-neighbor prediction rules and demonstrated that a single Softmax attention layer can function as a nearest-neighbor classifier.

[27] reproduce the phenomenon that the emergence of the induction attention heads associates with the emergence of the in-context learning [24] through designed experiments with input sequence consisting of several "item-label" pairs [5]. [27] construct a simple model to show the induction head can lead to in-context learning. Note that this "item-label" sequence task is similar to the anchor function of the classification task in this paper.

# 3 Definitions

In this section, we define the anchor function. We also propose a method for partitioning the dataset and define generalization on data and tasks.

## 3.1 Anchor function

We will give some examples of anchor functions, followed by a general definition. For concrete implementations and visual illustrations of these concepts, please refer to Fig. 4.1. The analytical formulations in this subsection (Eqs. (3.3), (3.4), and (3.6)) provide differentiable approximations that enable effective neural network learning, especially when anchor matches are imperfect.

### 3.1.1 One-anchor function

Consider one anchor function for identity learning $f(X) : \mathbb{R}^{n\times d} \rightarrow \mathbb{R}^d$, where $n$ is the number of tokens, and $d$ is the dimension of each token, $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$ and $\boldsymbol{x}_i \in \mathbb{R}^d$.

In each $X$, one and only one of $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{n-1}$ has to equal to a designated prompt anchor $\boldsymbol{a} \in \mathbb{R}^d$, and

$$f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \boldsymbol{x}_{i+1}, \quad \boldsymbol{x}_i = \boldsymbol{a}. \tag{3.1}$$

For example, when $d = 1$ and $a = 3, f(12, 33, 14, 3, 42, 22, 32, 20, 28) = 42$. This concept mirrors a task that finds out the name in the input sequence, for example, the output of "He is from China, his name is Mike, he likes reading." is "Mike" and the anchor is "name is". To use neural network to learn this function, each token is represented by a one-hot vector, i.e., $\boldsymbol{x} \in \{0, 1\}^d$, where $d$ is the vocabulary size.

The prompt anchor indicates a task or an operation for its next token. We can generalize the identity function to an arbitrary function, i.e.,

$$f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = g(\boldsymbol{x}_{i+1}), \quad \boldsymbol{x}_i = \boldsymbol{a}, \tag{3.2}$$

where $\boldsymbol{x}_i$ is called an anchor item and $\boldsymbol{x}_{i+1}$ is called a key item. This function can be further reformulated as

$$f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \frac{\sum_{j=1}^{n-1} g(\boldsymbol{x}_{j+1}) / \|\boldsymbol{x}_j - \boldsymbol{a}\|}{\sum_{j=1}^{n-1} 1 / \|\boldsymbol{x}_j - \boldsymbol{a}\|}. \tag{3.3}$$

Eq. (3.3) provides a continuous approximation of the anchor function, converting the discrete "exact match" condition to a weighted sum where tokens near $\boldsymbol{a}$ receive higher weights ($1/\|\boldsymbol{x}_j - \boldsymbol{a}\|$), enabling gradient-based optimization in neural networks.

In general, a one-anchor function satisfies the following form:

$$f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \frac{\sum_{j=2}^{n} g_j(\{\boldsymbol{x}_k | k \in I_j\}) / \|\boldsymbol{x}_j - \boldsymbol{a}\|}{\sum_{j=1}^{n-1} 1 / \|\boldsymbol{x}_{j-1} - \boldsymbol{a}\|}, \tag{3.4}$$

where $g_j$ is a function depending on index $j$, and $I_j$ is a set of indexes depending of $j$. In this equation (Eq. (3.4)), the index $j$ ranges from 2 to $n$ in the numerator and 1 to $n-1$ in the denominator to consistently cover tokens $\boldsymbol{x}_1$ to $\boldsymbol{x}_{n-1}$ (as $\boldsymbol{x}_{j-1}$ for $j = 2, \ldots, n$ corresponds to $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}$). This indexing ensures all anchor-influenced tokens are included, with $g_j$ depending on position $j$ and $I_j$ defining context-specific subsets. The formulation extends the discrete anchor concept to a general, position-aware approximation for arbitrary functions $g_j$, clarifying its role in enhancing model flexibility.

### 3.1.2 Two-anchor composite function

Consider one two-anchor composite function $f(X) : \mathbb{R}^{n \times d} \to \mathbb{R}^d$. The anchor set consists of a list of different tokens, i.e., $A = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_J\}$, and each token $a_k \in A$ corresponds to function $g(\boldsymbol{x}; a_k)$. In each input sequence $X$, one and only one pair of two consecutive elements belong to $A$ such as $\boldsymbol{x}_i, \boldsymbol{x}_{i+1} \in A$. Then,

$$f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = g\big(g(\boldsymbol{x}_{i-1}; \boldsymbol{x}_i); \boldsymbol{x}_{i+1}\big), \quad \boldsymbol{x}_i, \boldsymbol{x}_{i+1} \in A. \tag{3.5}$$

### 3.1.3 Forward-backward recitation anchor function

Consider one forward-backward recitation anchor function $f(X) : \mathbb{R}^{n\times d} \to \mathbb{R}^d$,

$$f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \begin{cases} \boldsymbol{x}_{i+1}, & \text{if} \quad \boldsymbol{x}_i = \boldsymbol{a}, \quad \boldsymbol{x}_n = \boldsymbol{x}_{i-1} \quad \text{(forward task)}, \\ \boldsymbol{x}_{i-1}, & \text{if} \quad \boldsymbol{x}_i = \boldsymbol{a}, \quad \boldsymbol{x}_n = \boldsymbol{x}_{i+1} \quad \text{(backward task)}. \end{cases}$$

This task also has an analytical formulation, i.e.,

$$f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = -\boldsymbol{x}_n + \frac{\sum_{j=2}^{n-1} (\boldsymbol{x}_{j+1} + \boldsymbol{x}_{j-1}) / \|\boldsymbol{x}_j - \boldsymbol{a}\|}{\sum_{j=2}^{n-1} 1 / \|\boldsymbol{x}_j - \boldsymbol{a}\|}. \tag{3.6}$$

Eq. (3.6) approximates the recitation anchor function through a weighted average. The sum runs from $j = 2$ to $n - 1$ to focus on interior tokens (avoiding boundary effects), with $-\boldsymbol{x}_n$ handling sequence-end constraints. This differentiable formulation supports stable learning of recitation tasks, demonstrating how analytical expressions implement discrete anchor concepts. This anchor function simulates the recitation of a previous sentence or a following sentence with the hint of the current sentence.

### 3.1.4 General anchor function

A general anchor function with an anchor set $A$ can be formulated in the following form:

$$f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = g(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n; \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \cap A), \tag{3.7}$$

where $g$ is a function of a subset of $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, where $g$ depends on the anchor terms $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \cap A$, and the subset items are called key items.

This general formulation encompasses the preceding specific cases:

- For one-anchor functions (Section 3.1.1), $A = \{\boldsymbol{a}\}$ and $g$ depends solely on the adjacent key item $g(\boldsymbol{x}_{i+1}; \boldsymbol{a})$.
- For two-anchor composites (Section 3.1.2), $A = \{\boldsymbol{a}_i, \boldsymbol{a}_j\}$ with nested $g$ operations $g(g(\boldsymbol{x}_k; \boldsymbol{a}_i); \boldsymbol{a}_j)$.
- For forward-backward recitation (Section 3.1.3), $A = \{\boldsymbol{a}\}$ with conditional $g$

$$g = \begin{cases} \boldsymbol{x}_{i+1}, & \text{if} \quad \boldsymbol{x}_n = \boldsymbol{x}_{i-1}, \\ \boldsymbol{x}_{i-1}, & \text{if} \quad \boldsymbol{x}_n = \boldsymbol{x}_{i+1}. \end{cases}$$

Thus, $g(\cdot; \{\boldsymbol{x}_i\} \cap A)$ provides a unified framework where the functional dependence is determined by both key items and the specific anchor combination in the input.

## 3.2 Training and test data

We generate synthetic datasets for evaluating anchor functions. Each input sequence consists of $n$ tokens, where each token is represented as a one- or two-digit positive integer ($x_i \in \{1, 2, \ldots, 99\}$). The data generation process ensures:

- **Anchor presence**: Each sequence contains exactly one anchor-key pattern satisfying the task requirements defined in Section 3.1 (e.g., for one-anchor functions, exactly one $\boldsymbol{x}_i = \boldsymbol{a}$ appears in positions 1 to $n-1$).
- **Random generation**: Non-anchor tokens are randomly sampled from a uniform distribution over $\{20, 21, \ldots, 99\}$.
- **Controlled generation**: Anchor positions are randomly assigned while maintaining task constraints (e.g., consecutive anchors for two-anchor functions).

This synthetic approach enables reproducible evaluation of generalization behavior. For example, a one-anchor identity learning sequence with $n = 5, \boldsymbol{a} = 3$ might be generated as $[12, \mathbf{3}, 42, 7, 18]$, where 42 is the key item.

Due to the complexity of natural language, cleanly separating training and test data is often challenging. The idealized anchor function appears to be capable of accomplishing the aforementioned task. However, a straightforward division based on data ranges proves to be impractical. To illustrate, consider a scenario where the range of key items in the training set is denoted as $[a, b]$, while in the test dataset, it is represented as $[b+1, c]$. The encoding of data within the interval $[b+1, c]$ is not learned during the neural network training process. As a result, the neural network fails to produce the key item output for the test dataset.

In this section, we propose three methods to separate training and test data, as shown in Fig. 3.1.

### 3.2.1 Modular-residue data separation

This separation method addresses a fundamental challenge in evaluating compositional generalization: preventing models from relying on positional memorization rather than learning core mapping functions. In identity mapping tasks, where the relative positions between anchors and key items determine task semantics, conventional data splits risk containing identical (value, position) pairs in both training and test sets. Such overlap would make it impossible to distinguish whether a model has genuinely learned the identity function or simply memorized specific positional mappings.

Consider a task with an input sequence of length $n$. Define $\Gamma_i$ as a set depending on the position $i$. For example, $\Gamma_i = \{i-2\}$ for the case where key items can not be placed in the first position. Modular-residue data separation is as follows.

For an input sequence of the training dataset, if the token of the $i$-th position is a key item, then, a number $x$ can be placed in the $i$-th position of such input sequence only when $\mathrm{mod}(x, n-1) \notin \Gamma_i$.

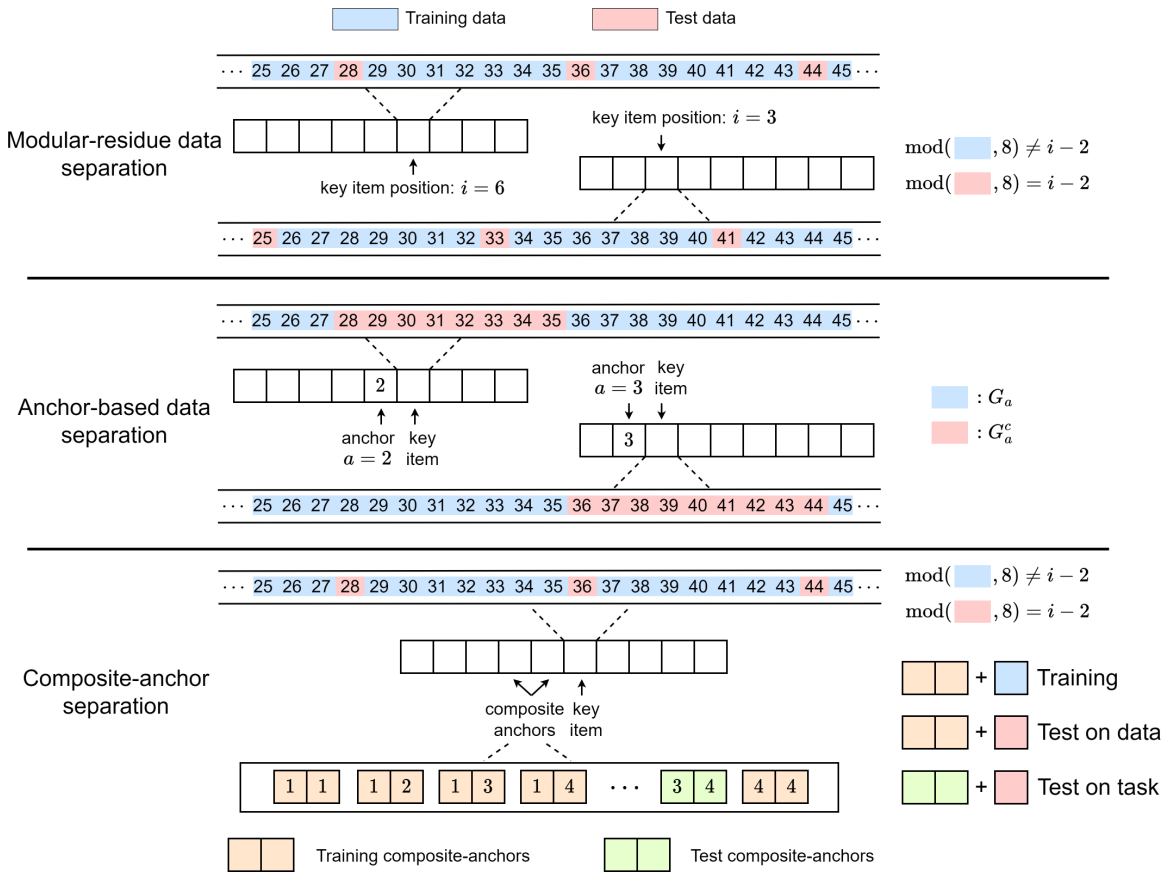



Figure 3.1: Schematic diagram of three ways to split the training set and the test set.

For an input sequence of the test dataset, if the token of the $i$-th position is a key item, then, a number $x$ can be placed in the $i$-th position of such input sequence only when $\mathrm{mod}(x, n-1) \in \Gamma_i$.

This formulation guarantees complete separation between training and test distributions at the (value, position) level. Crucially, it requires models to learn the underlying identity mapping function rather than memorizing specific token-position combinations, thereby enabling unambiguous evaluation of true reasoning capabilities. The position-dependency of $\Gamma_i$ reflects the task structure where anchor-key relationships are inherently position-sensitive.

### 3.2.2 Anchor-based data separation

Consider a problem with multiple anchors. Denote the anchor set as $A = \{a_i\}_j^J$. Take an example that in each input sequence, one and only one token belongs to the anchor set. Anchor-based data separation is as follows.

In the training data set, key items of anchor $a_i$ belong to a set denoted as $G_{a_i}$ and its output token belongs to a set denoted as $G^o_{a_i}$. Denote the set of all tokens as $I$ and the set of all possible output tokens as $I^o$. These sets satisfy the following requirements: $G_{a_i} \subsetneq I$ and $G^o_{a_i} \subsetneq I^o$, $I = \cup G_{a_i}$ and $I^o = \cup G^o_{a_i}$. The key items for $a_i$ in a test sequence should belong to $G^c_{a_i} = I \backslash G_{a_i}$.

#### 3.2.3 Composite-anchor separation

Consider an anchor set with multiple anchors. In each input sequence, there exists a composition of certain anchors. The compositions of anchors in the training set and in the test set do not overlap.

### 3.3 Generalization

The division of the dataset naturally leads to two concepts of generalization: generalization on data and generalization on tasks. The first type describes the model's ability to generalize to seen tasks and unseen data, while the second type pertains to its ability to generalize to unseen tasks. This distinction is operationally defined by whether test data contains anchor combinations identical to training (data generalization) or novel anchor combinations (task generalization).

**Generalization on data.** Generalization on data primarily relies on the test set, which is divided according to key items. In this test data, the anchor function (i.e., the task) has already been encountered in the training set. This form of generalization is based on data separation, such as the case generated through modular-residue and anchor-based data separation.

**Generalization on task.** Generalization on task primarily depends on the test set, divided based on the anchors. Although some anchor combinations in the test data are new (not present in the training set), the model can infer unseen composite anchor functions using the existing composite anchors from the training set.

### 3.4 Loss function

In this work, we use a transformer architecture, which takes an input sequence of length $n$, subsequently yielding an output matrix $X^{(\text{out})} \in \mathbb{R}^{n\times d}$, where $d$ is the size of the dictionary. For the $i$-th input sequence, the loss function is the cross-entropy only for the output of the last token after softmax (denoted as $\mathbf{x}_n^{(\text{out},i)} \in \mathbb{R}^d$). The total loss over the entire training dataset is computed as

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{d} \boldsymbol{y}_c^{(i)} \log\left(\mathbf{x}_{n,c}^{(\text{out},i)}\right),$$

where $\boldsymbol{y}^{(i)}$ is a one-hot label for the $i$-th sequence and $N$ represents the size of the training dataset.

# 4 Tasks and results

In this section, we apply transformer models in various tasks (Fig. 4.1) to simulate real-world scenarios and delve into certain mechanisms underlying them. Without specification, we employ a 4-layer decoder-only transformer network to learn tasks and we set the sentence length to 9 and the dictionary size to $d = 200$ for each task. Each decoder layer comprises four heads. Please refer to the Appendix A for detailed settings. The loss and accuracy for each task during the training are summarized in Fig. D.1. A multi-layer transformer neural network can learn and generalize all listed examples.

## 4.1 Identity learning task

We begin by examining the identity learning function, which simulates a task to find out the name in the input sequence. For example, the output of "He is from China, his name is Mike, he likes reading." is "Mike" and the anchor is "name is".

The identity learning function is defined in Eq. (3.1), where the anchor item serving as a prompt occurs only once in the input sequence. The function's output is the digit following that specific anchor. In our experiments, we use "3" as the anchor term. Meanwhile, we study the mechanism by which the transformer model has good generalization ability on the identity learning task in Section 5.

We compared generalization capabilities of fully-connected neural networks (FNNs), long short-term memory networks (LSTM; plural LSTMs), and transformers in the identity learning task. The respective parameter counts for these models are 416 M, 5.3 M, and 5.6 M. Please refer to the appendix for detailed setups for the three models. As shown in Fig. 4.2, the transformer demonstrates excellent generalization performance with just 600 training data points, while LSTM requires around 2200 training data points to achieve a similar level of generalization. FNN, on the other hand, fails at generalization. This phenomenon, along with some existing works, highlights the significant advantage of transformer models over FNN models in logical reasoning [37].

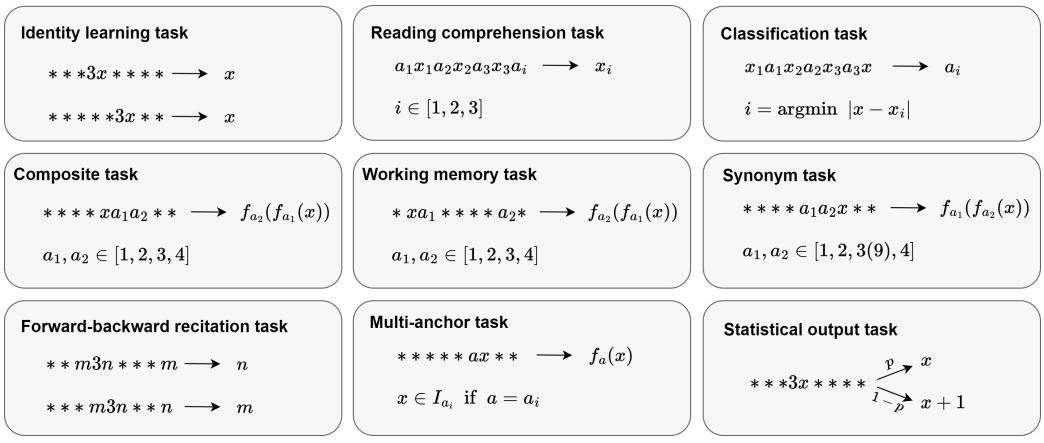



Figure 4.1: Task examples of anchor functions.

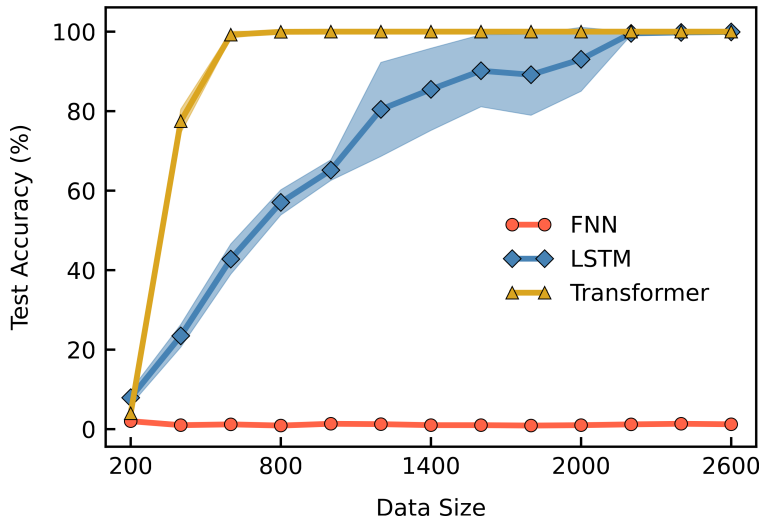



Figure 4.2: Performance comparison of FNN, LSTM, and transformer. For each experimental setting, we used 10 random seeds for the experiment. Each experiment was trained for 4000 epochs. The scatter points are the average of the ten experimental results, and the shaded parts are the maximum and minimum values.

## 4.2 Reading comprehension task

In daily reading comprehension, one usually identifies the anchors in the question first and then searches the similar anchors in the given text. The answer can often be found in the content near the anchor. We set the following task to simulate this reading comprehension task. We define distinct anchor values $a_i \in \{1, 2, \ldots, 8\}$, and item values $x_i \in [20, 100]$. The input sequence follows the pattern $(a_1, x_1, a_2, x_2, a_3, x_3, a_4, x_4, a)$, where $a_i$ is an anchor, $a_i \neq a_j$ for $i \neq j$. The output label is $x_k$ where $k$ satisfies $a_k = a$. In this task, the division of the training set and test set is done through the "Modular-residue data separation" method. We use a total of 1000 data points for training, with a batch size of 50, and train for 1000 epochs.

## 4.3 Classification task

The classification task is a variation of the reading comprehension task, focusing on content that is similar rather than identical to the anchors. In this task, we define distinct anchor values $a_i \in \{1, 2, \ldots, 8\}$, and item values $x_i \in [20, 100]$. The input sequence follows the pattern $(x_1, a_1, x_2, a_2, x_3, a_3, x_4, a_4, x_5)$, where $a_i$ is an anchor, $a_i \neq a_j$ for $i \neq j$. The output label is $a_k$ where $k = \text{argmin}_{i \in \{1,2,3,4\}} |x_i - x_5|$. In this task, the division of the training set and test set is done through the "Modular-residue data separation" method. We use a total of 30000 data points for training, with a batch size of 50, and train for 4000 epochs.

As shown in Fig. 4.3, we visualize the token embedding by t-SNE [32] after training. Obviously, the embedding captures the order of tokens, which is crucial in this classification task.

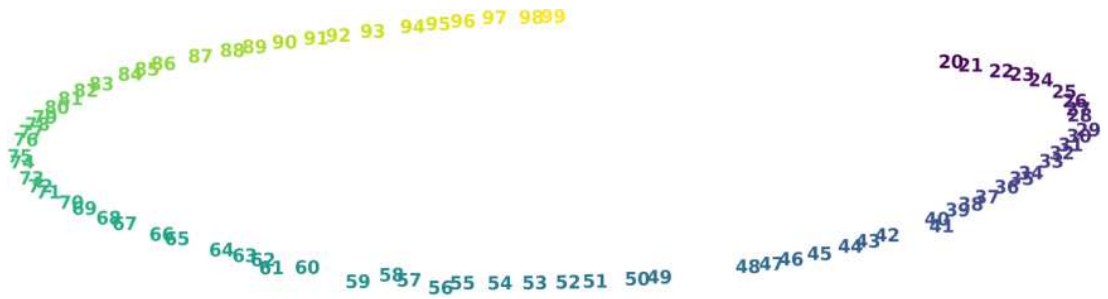


Figure 4.3: The t-SNE distribution of token embeddings after training for numbers from 20 to 99.

## 4.4 Composite task

The composite task is a natural extension of one-anchor tasks. In each input sequence, there are always two anchors to indicate a composite function. A fundamental question arises: within the composite task, can the model learn the function of each individual anchor that constitutes the composite function? In this task, we separate the data by both modular-residue data separation and composite-anchor separation. Therefore, we can study the generalization of transformer on data and tasks. We also study how model size affects the performance.

Initially, we define four anchors, each representing a specific operation. These anchors are then paired up to create 16 distinct composite anchor functions. 15 pairs are selected to construct the training set. The model's generalization ability is tested across all 16 operations. Notably, the model's generalization on the 15 composite operations used for training reflects its ability to generalize on data, while its generalization on the composite operations absent from the training set reflects its ability to generalize on tasks.

As shown in the left in Fig. 4.4, we denote $f_a$ for the function of anchor $a$ and combine each single anchor to obtain the two-anchor composite function $f_{a_1,a_2}$ of the anchor pair $\{a_1, a_2\}$. The gray anchor pair in the "Composite output" part does not appear in the training set. In the right in Fig. 4.4, we illustrate the model with 10 layers after training via three input examples. We found the network learns a specific composite function. For the first anchor $a_1$, network outputs $(f_{a_1} f_{a_1})(x)$, where $x$ is the key item. For the second anchor, network outputs $(f_{a_2} f_{a_1}^{-1})(f_{a_1} f_{a_1})(x)$, which cancels one operation of $f_{a_1}$.

We found model size can significantly affect the output pattern after training. In the case of the model with 10 layers, as shown in the right in Fig. 4.4, after the input of the second anchor, the network outputs the correct answer and keeps broadcasting the correct answer for the subsequent input tokens. To characterize the output pattern for a specific composite of anchors, we define the mapping sequence for an input as follows.

**Definition 4.1** (Mapping Sequence)**.** *Consider a model $f_{\boldsymbol{\theta}}(\cdot)$ and an input sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, where $i$ denotes the index of the key item, and $i+1$ and $i+2$ represent the anchor indexes. The model output is $f_{\boldsymbol{\theta}}(x) = (f_{\boldsymbol{\theta}}(x_1), \ldots, f_{\boldsymbol{\theta}}(x_n))$. Then, define the sequence $(f_{\boldsymbol{\theta}}(x_{i+1}) - x_i,\ f_{\boldsymbol{\theta}}(x_{i+2}) - f_{\boldsymbol{\theta}}(x_{i+1}), \ldots, f_{\boldsymbol{\theta}}(x_j) - f_{\boldsymbol{\theta}}(x_{j-1}))$ as a mapping sequence of the model $f_{\boldsymbol{\theta}}$, where $f_{\boldsymbol{\theta}}(x_k) = f_{\boldsymbol{\theta}}(x_{k-1})$ for any $k > j$ and $f_{\boldsymbol{\theta}}(x_j) \neq f_{\boldsymbol{\theta}}(x_{j-1})$.*

As shown in Fig. 4.5(a), for the case of the model of 10 layers, there is only one mapping sequence for each composite of anchors. However, when the model size is smaller, it shows several different mapping sequences for each composite of anchors. This result depending on the model size is an interesting phenomenon for further study. In addition, the test accuracy on task is usually smaller than the test accuracy on data (always achieves 100%) after training. This suggests the generalization on task is more difficult. To verify this, we display a learning process in Fig. 4.5(b). The training accuracy and the test accuracy on data achieve 100% much earlier than the test accuracy on task.

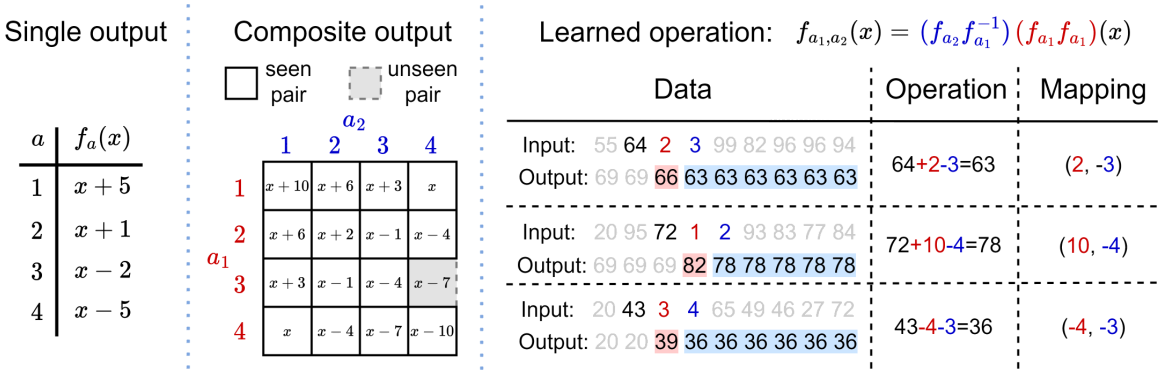



Figure 4.4: Single output: Function mapping of one anchor. Composite output: The two-anchor composite function mapping obtained after the composition of two one-anchor functions. Learned operation: The mapping operation of the composite function actually learned by the model. The examples shown represent the actual outputs of the model for the given inputs, and the corresponding operations on the examples support the learned operation of the model.

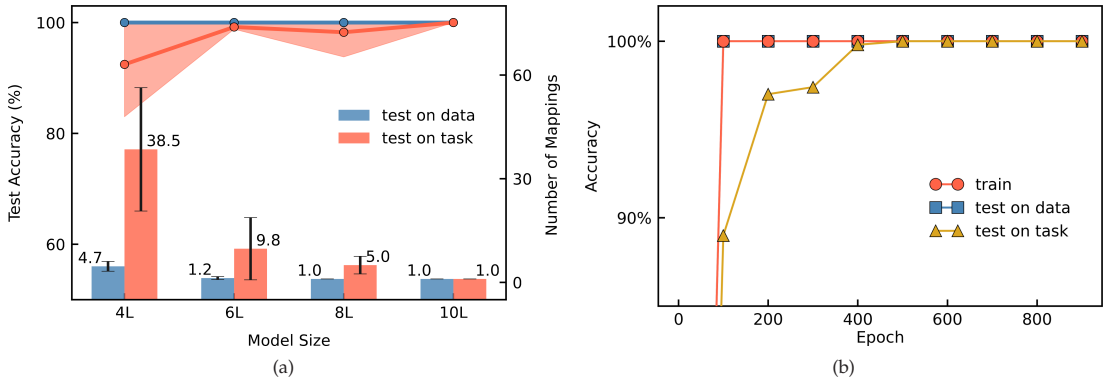



Figure 4.5: (a) The relationship between test accuracy (left ordinate, curve) and the number of mapped sequences (right ordinate, bar) with the model size, where "L" represents the layer. The blue color reflects the generalization ability on data, while the red color reflects the generalization on task. Each configuration is tested five times independently, and the mean is taken. The shaded area and error bars respectively represent the variance in test accuracy and the number of mapped sequences. (b) The training accuracy and test accuracy of the composite task. The convergence speed of the model's generalization on the data is much faster than its convergence speed on the task.

### 4.5 Working memory task

The composite task demonstrates the model's ability to learn a single anchor operation from composite anchors. The second anchor processes the output from the first anchor instead of directly processing the key item. This sequential processing necessitates working memory capabilities in the model if there are some irrelevant items between the two anchors.

Specifically, we let the first anchor $a_1 \in \{2,3\}$ and the second anchor $a_2 \in \{4,5\}$, with their corresponding mappings illustrated in Fig. 4.6. Notably, the model observes only the final output and not the intermediate computations. Therefore, as long as the final output of two composite anchors is the same as the given labels, the network can learn a different function for a single anchor operation, compared with our given operation for the anchor.

As depicted in Fig. 4.6, the first anchor's output is the intermediate result via the operation of the first anchor on the key item. The model memorizes this intermediate result by keeping out this value for the input of irrelevant tokens. Once the model sees the second anchor, the intermediate result is no longer useful and the model outputs the final correct answer.

### 4.6 Synonym task

The synonym task is an extension of the composite task, aimed at studying the capacity of language models to learn the synonymous relationship between two anchors. The composite anchors in the original dataset are taken from: $\{(1,9),(2,9),(9,1),(9,2),(1,2),(1,4),(2,1),(4,1),(2,4),(4,2)\}$. The number of the training data set for each composite anchor is the same. However, the transformer network struggles with learning composite anchors such as $(4,9)$ and $(9,4)$ under the above setup for different total training data sizes, as shown by the blue curves in Fig. 4.7(a).

We set an anchor "3" that performs the same function as anchor "9". To see whether

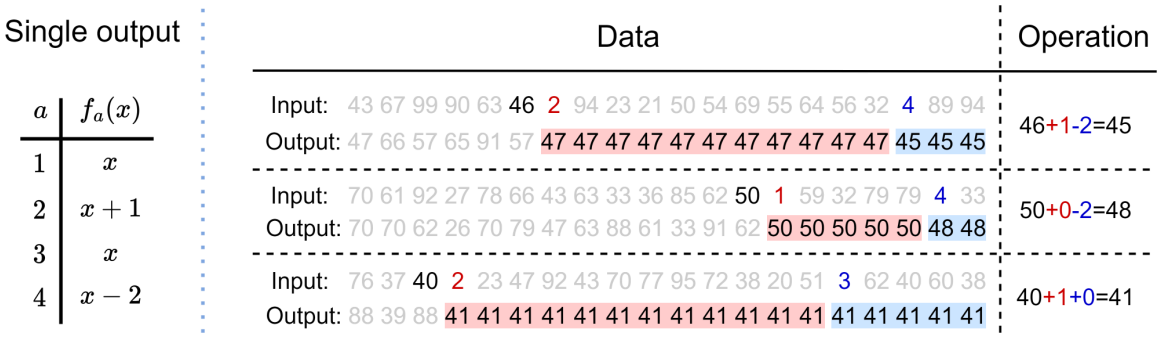



Figure 4.6: Single output: Mapping function for one anchor, where $a_1 \in \{1,2\}, a_2 \in \{3,4\}$. Data: Input sequence and its corresponding output from the model. The red shadow represents the working memory for $a_1$, and the blue part corresponds to the working memory for $a_2$. Operation: The learned mapping for the anchors by the model.

transformer networks can learn the function of anchors $(4,9)$ and $(9,4)$ through the composite anchors $(4,3)$ and $(3,4)$, we add the data with synonym anchor "3" to the training set. Specifically, we integrate composite anchors $\{(1,3),(3,1),(2,3),(3,2),(4,3),(3,4)\}$ into the original dataset. In the case with a total number of training data as $M$, the number of training data for composite anchors that contain "9" (such as $(1,9)$) is fixed as 100, while for other composite anchors is the $(M-400)/12$. As shown by the red curves in Fig. 4.7(a), the network can accurately predict the output of $(4,9)$ and $(9,4)$ as long as the data size is larger than a certain value. In Fig. 4.7(b), we display the learning process for cases of two data sizes. Obviously, the learning of $(4,9)$ and $(9,4)$ is slightly later than $(4,3)$ and $(3,4)$. These results demonstrate that introducing synonyms significantly impacts the transformer's task generalization.

## 4.7 Forward-backward recitation task

Usually, one finds that reciting the next line of a poem (forward task) is much easier than reciting the previous one (backward task). Empirically, LLM is also found to be easier for forward tasks than backward tasks. For example, we use GPT4 to test the forward-backward recitation task with 20 poems. In the task of forward recitation, GPT4's accuracy is 95%, while the accuracy of the backward recitation is only 5%. Based on this observation, we design an idealized forward-backward recitation task to study this asymmetry of LLMs. An input sequence follows the pattern "$(*,*,m,3,n,*,*,*,n)$" or "$(*,*,m,3,n,*,*,*,m)$", where "3" is the anchor, and others are tokens randomly selected from $[20,100]$, "$*$" indicates that the token in this position is irrelevant, and "$m,3,n$" can be any position. The output of "$(*,*,m,3,n,*,*,*,n)$" is "$m$", while the output of "$(*,*,m,3,n,*,*,*,m)$" is "$n$". As shown in Fig. 4.8, the transformer network's general-

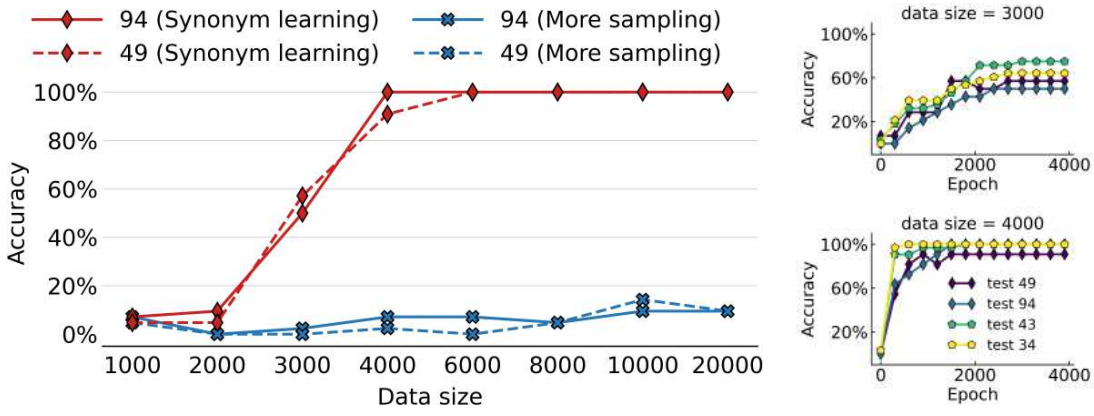



Figure 4.7: (a) Compare the impact of two types of data enhancement methods on task generalization. These types of data enhancement methods include adding synonyms (red) and sampling more data with the original composite anchors (blue). (b) The test accuracy of transformer predictions on data with unseen anchors in the synonym task. The composite anchors $(4,9)$ and $(9,4)$ are unseen anchors. training data size for the upper and lower cases are 3000 and 4000, respectively.

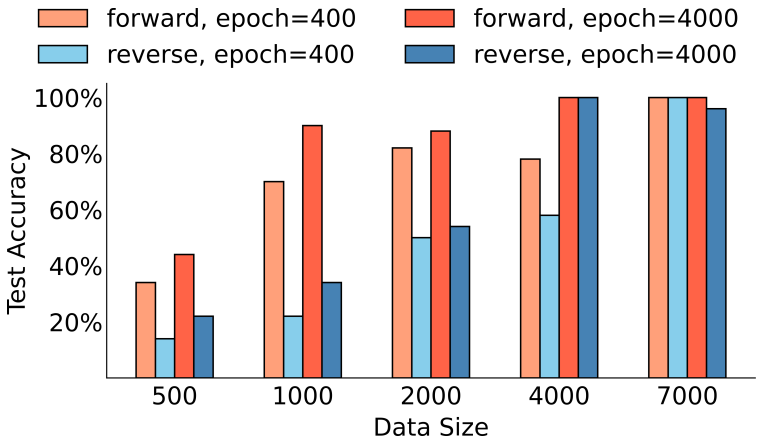



Figure 4.8: Accuracy of forward prediction and backward prediction in forward-backward recitation task changes with training data size. The figures show the accuracy of the test data when training to 400 epochs and 4000 epochs respectively.

ization ability in forward tasks is consistently superior to that in backward tasks.

## 4.8 Statistical output task

Statistical output is a common scenario, i.e., the output of a sequence follows a probability distribution. To simulate this scenario, we design a task based on the identity learning anchor function.

We examined two types of dataset construction:

- The dataset $S$ is built in the same way as the identity learning task and replicated five times, modifying the label of the final copy to $x+1$. These five datasets constitute the first type of fairness learning dataset.
- The dataset $S$ is built in the same way as the identity learning task and randomly changed the labels of 20% of the data to $x+1$. The newly obtained dataset is called the second type of fairness learning dataset.

As shown in Fig. 4.9(a), for the first type of data, the training accuracy can only achieve 80%. This is because we only consider the output with the largest probability, while the output for an input is statistical in the training dataset. For the data with label $x$, the network can achieve 100% while 0% for the data with label $x+1$. For the second type of data, as shown in Fig. 4.9(b), the probability of output $x$ achieves 80% while 20% for output $x+1$.

Next, we examine if the network learns the probability distribution of output. For input, we can record the probability of output $x$ and $x+1$, respectively, i.e., softmax of the output layer. As shown in Fig. 4.9(c), for the first type of data, as the epoch increases, the probability of output $x$ increases towards 80% while $x+1$ towards 20%, both of which satisfy the experimental setting.

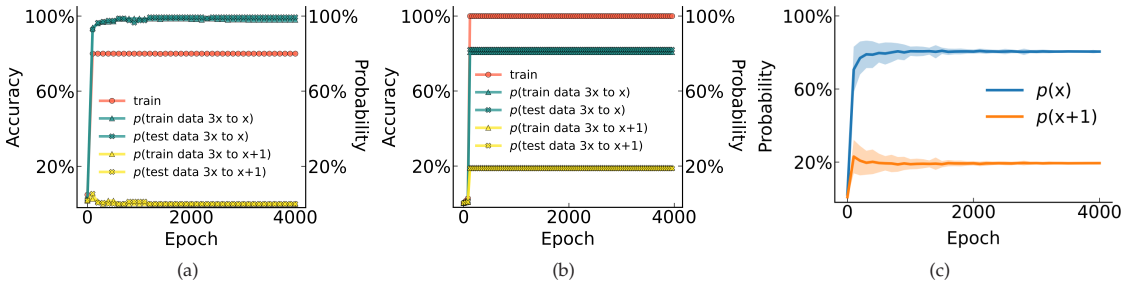



Figure 4.9: (a, b) The accuracy of transformer predictions in the statistical output task. The transformer is trained with the first type of dataset (a) and the second type of dataset (b). (c) For the first type of dataset, the probability of token "$x$" or "$x+1$" after the projection layer.

## 4.9 Multi-anchor task

In the case of modular-residue data separation, it only ensures different positions for the pair of anchor and key items i.e., $(a, x)$, in the training and test sets. One might argue the network only learns shift invariation but not the operation. To verify the learning of operation, we design the following multi-anchor task.

We utilize ten distinct anchors $a \in \{1,2,3,4,5,6,7,8,9,10\}$ and define the anchor function $f_a(x) = x + a$, where $x$ is the key item next to the anchor. For the test dataset, the key item for anchor $a$ belongs to $G_a^c = [12+8a, 12+8(a+1)-1]$, and for the training set, the key item for anchor $a$ belongs to $G_a = [20,100]\backslash G_a^c$, i.e., anchor-based data separation. Our focus is on whether, after training, the transformer's predictive ability can generalize to sentences where the pair of anchor and key item is unseen in the training.

As shown in Fig. 4.10, in most cases, the model exhibits good generalization capabilities on anchor-based data, i.e., the network learns the operation rather than shift-invariance. For the cases near the boundary (anchor "1", "2", "10"), due to the construction method,

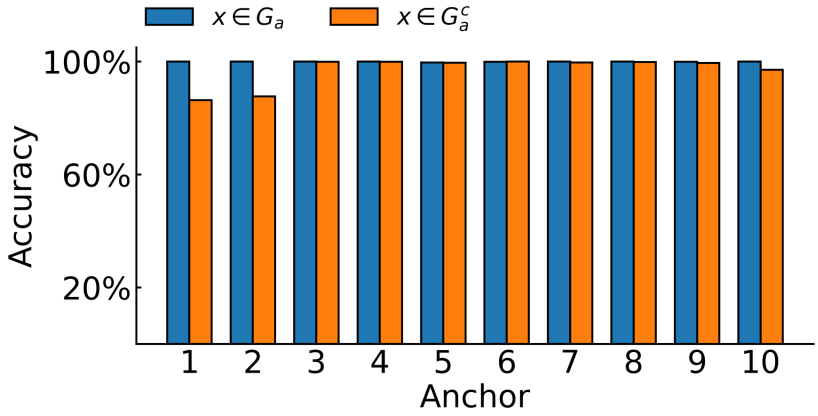



Figure 4.10: Accuracy of multi-anchor task after training. We examined the predictive ability of transformer on data sets constructed with different anchors. The test set is $G_a^c = [12+8a, 12+8(a+1)-1]$, and the training set is $G_a = [20,100]\backslash G_a^c$.

the frequency of their output tokens is smaller than other anchors, which may be the reason why the learning boundary anchors are more difficult.

# 5 Mechanism study of the identity learning task

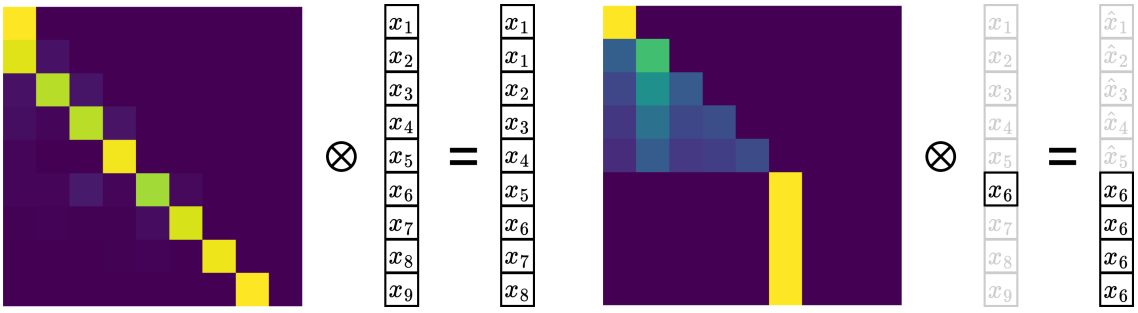
In this section, we first provide a brief explanation of the mechanism of the identity learning task by identifying two key operations in a two-layer model, i.e., shift and broadcast. We then conduct a series of ablation experiments to explore the role of each structure in the transformer and find a very simplified two-layer model that can accomplish the identity learning task. The operations of shift and broadcast also play key roles in this simplified model. Finally, we show these two operations are also common in LLMs and have a short discussion. A very detailed discussion of the mechanism for the identity learning task can be found in Appendix B.

## 5.1 A brief explanation of a two-layer model

In the task of identity learning, the attention maps of the two-layer model primarily serve two purposes: shift and broadcast. The first layer's attention map performs a shift operation, as illustrated in Fig. 5.1(a), the sequence $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$ is shifted to $(x_1, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ through the first layer's attention map. Based on the residual connection, the shifted vector and the unshifted one is fused, thereby establishing the integration of information between the anchor item and the key item. It is noteworthy that the operation is a one-position shift, due to the fact that the key item is designed to be the next token of the anchor. We will discuss more on how the data structure affects the shift operation.

Once the information of the anchor and the key item is fused, the subsequent fully-connected structure will render the correct label based on this fused information.

The second layer's attention map executes a broadcast operation. As illustrated in Fig. 5.1(b), the sequence $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$ is transferred to $(\hat{x_1}, \hat{x_2}, \hat{x_3}, \hat{x_4}, \hat{x_5}, x_6, x_6, x_6, x_6)$, where $\hat{x_i}$ is the linear combination of $\{x_1, \ldots, x_i\}$ but is irrelevant to obtaining




(a) First layer, shift operation

(b) Second layer, broadcast operation

Figure 5.1: Attention maps of the first and second layer.

the correct label. This broadcast operation transfers the correct label to the last position, namely the output position. The first layer's attention map is mainly based on the position embedding and the second layer's attention map is mainly based on the positions of anchors in the input sequence.

This example illustrates two important operations that attention structure provides in language models, i.e., shift and broadcast. We also find that these two operations are also common in LLMs (see experiments in later sections). Therefore, the anchor function can serve as a type of benchmark function for understanding transformer-based language models.

To realize the attention maps with shift and broadcast operations, one possible way is to tune vector orientations in high-dimensional space. Two random vectors in a high-dimensional space are orthogonal with high probability. For the shift attention, the training needs to tune the $i$-th row of matrix $Q$ to parallel with the $(i-1)$-th row of matrix $K$. For the broadcast one, the training needs to identify the anchor token. The fully-connected network non-linearly transforms all tokens to parallel ones (called non-interested tokens for convenience), except for the interested one that has combined the anchor token "3" and the key token "$x$". Rows of non-interested tokens in the matrix $Q$ (or $K$) are parallel and the interested one is orthogonal to others. For non-interested token positions, the direction of each row of $Q$ is parallel but opposite to the corresponding one of $K$, while for the interested position, parallel with the same direction. Then, the element in attention that does not contain the interested position will be very negative, leading to the broadcast operation. Please refer to the Appendix B for a detailed analysis of the mechanisms of the two-layer transformer.

The understanding of the mechanism behind identity mapping tasks above is built upon extensive experiments and visual analysis. The low training cost and clear data structure of anchor functions greatly facilitated our research into the implementation mechanisms behind logical rules.

## 5.2 Data structure's impact on the shift attention map

The attention map in the aforementioned model is heavily influenced by structural features inherent in the training data, particularly the adjacency of key items with anchors in the identity learning task. In this section, we investigate the influence of data structures on the attention map. We generate data for the identity learning task using two anchors, namely "3" and "4". For the data corresponding to anchor "3", the key item is the digit following "3", and the task is represented as $(\ldots, 3, x, \ldots) \to x$. For the data corresponding to "4", the key item is the digit located two positions after "4", and the task is represented as $(\ldots, 4, *, x, \ldots) \to x$. This investigation is conducted within a transformer model comprising two layers, each containing two heads.

Figs. 5.2(a) and 5.2(b) illustrates the attention maps of the two heads in the first layer of the model. Both heads perform shift operations, with the first head, shown in Fig. 5.2(a), shifting by one position, aligning with the task corresponding to the anchor "3", while the other head, shown in Fig. 5.2(b), shifts by two positions, aligning with the anchor "4". This phenomenon is caused by the characteristics of the training data and the positional

(a) First layer, first head (b) First layer, second head (c) Second layer, anchor "3" (d) Second layer, anchor "4"

Figure 5.2: (a, b) Attention maps of the two heads of the first layer. (c, d) Attention maps of two heads of the second layer for the input sequence with anchor "3" and "4", respectively.

embedding obtained during training. During the testing phase, regardless of the input test sequence, these two attention maps will remain stable.

For the attention maps of different heads in the second layer, they perform broadcast operations. Figs. 5.2(c) and 5.2(d) illustrates the attention map of the second layer head, taking "3" and "4" as anchors, respectively (only the first head is shown, and the other head is similar). When the positions of two anchors are the same, the positions of the key items corresponding to the two anchors are different. Therefore, the positions performing the broadcast operation for the attention map of anchor "4" are one position later than anchor "3".

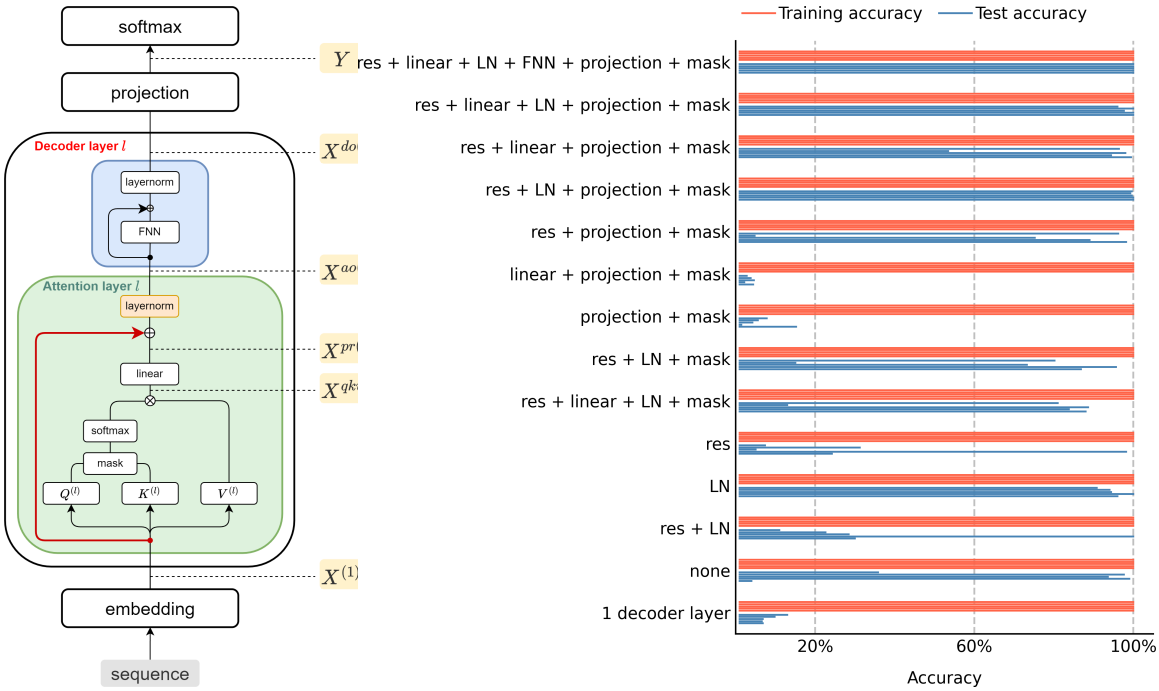
## 5.3 Ablation experiment

In the following, we empirically study how much we can simplify the decoder-only network structure, followed by analyzing a simplified model's mechanism in the identity learning task, which may be more accessible for future theoretical study.

A common block of a transformer network is shown in Fig. 5.3(a). The detailed operation of each component is illustrated in Appendix B.1. To find a model that can accomplish the anchor function but contains as few components as possible, we conduct ablation experiments of different components for the identity learning task with modular-residue data separation.

Empirically, we find that the one-layer decode-only structure can not generalize at all for the identity learning task, while the two-layer one generalizes well. The reason why the one-layer structure fails is that each token is solely processed. Therefore, the fusion of different tokens cannot be done via the one-layer structure. If we manually shift vector $V$ as the shift operation, then the network can easily fit the identity learning task.

We then conduct ablation experiments on various components of a two-layer model: "res" indicates the residual connection, "linear" indicates the linear transformation after attention operation, "LN" indicates the layer normalization, "FFN" indicates the feedforward neural network, "projection" is a linear transformation of the output of the decoder before the final softmax, "mask" indicates that each token can only see its previous tokens but not subsequent ones. It is worth mentioning that when we remove certain modules,




(a) (b)

Figure 5.3: (a) Transformer structure used in experiments. (b) Ablation experiments. For the first 3 and last 2 experiments, the row vector dimension of $X^{(l)}$ is 400, and the row vector dimension of $Q^{(l)}, K^{(l)}, V^{(l)}$ is 64. For others, all of those dimensions are 201. For each experiment setting, we run the experiment 5 times with different seeds for 4000 epochs to eliminate the impact of randomness.

the dimensions of some matrices within the network need to be adjusted accordingly to ensure that the model is well-posed. For example, when we delete the linear operation, we need the row vector dimension $d_v$ of $V^{(l)}$ to be the same as the row vector dimension $d_m$ of the hidden state; when we delete the projection layer, the hidden state row vector dimension $d_m$ needs to be the same as the dictionary size $d$. Beyond that, we adjust $d_m, d_k$, or $d$ so that the model used in each experiment has roughly the same amount of hyperparameters.

For each experiment setting, we run the experiment five times with different random seeds. As shown in Fig. 5.3(b), the first experiment contains all regular components. For all five random seeds, the training accuracy and the test accuracy achieve 100%. Therefore, the five mini-bars together look like a large bar. In the second experiment, the FFN is removed. The training accuracy can always achieve 100%, but the test accuracy has slight variation across five trials.

It is interesting to see that the effect of different components is highly nonlinear. For example, the experiment of "linear+projection+mask" generalizes very badly for all five trials, but when all examined components are removed, the network can generalize better, as shown by the experiment of "none", which can be 100% although with large fluctuation. These experiments suggest there are many different mechanisms to learn an anchor function.

### 5.4 A mechanism of a two-layer simplified model

In this section, we employ a simplified two-layer model, which removes structures including masking, layer normalization, residual connection, linear transformation, and fully-connected networks, i.e., "none" in Fig. 5.3, to investigate the mechanisms of the identity learning task. We begin by providing the definition of the simplified model.

Let $X^{(1)} \in \mathbb{R}^{n \times d_m}$ represent the input sequence after word embedding and position embedding, i.e., $X^{(1)} = X^{\text{em}} + X^{\text{pos}}$. We choose $d_m$ to be equal to the dictionary size $d$ to simplify the linear operator in the transformer and delete the projection layer. The final output is calculated as follows:

$$\text{Attn}^{(1)} = \text{softmax}\left(\frac{X^{(1)}W^{Q(1)}W^{K(1)T}X^{(1)T}}{\sqrt{d}}\right), \quad X^{(2)} = \text{Attn}^{(1)}X^{(1)}W^{V(1)},$$
$$\text{Attn}^{(2)} = \text{softmax}\left(\frac{X^{(2)}W^{Q(2)}W^{K(2)T}X^{(2)T}}{\sqrt{d}}\right), \quad X^{(\text{out})} = \text{Attn}^{(2)}X^{(2)}W^{V(2)}.$$

It can be expressed simply as

$$X^{(\text{out})} = \text{Attn}^{(2)}\text{Attn}^{(1)}X^{(1)}W^{V(1)}W^{V(2)}.$$

It is noteworthy that $\text{Attn}^{(l)}$ incorporates information from $X^{(l)}$. Therefore, the aforementioned formula does not represent a straightforward linear transformation. The mechanistic explanation of the model is depicted in Fig. 5.4, where the first-layer attention map shifts the information of the key item "$x$" to the position of anchor item "3" and broadcasts the information of the anchor item to other locations beyond it. Consequently, there are only two types of matrix row vectors input to the second layer: vectors with the anchor item information and vectors with the key item information. This structure also facilitates the construction of the second-layer attention map $\text{Attn}^{(2)}$. The attention map $\text{Attn}^{(2)}$ performs a broadcast operation, i.e., broadcast the vector with the key item information to all locations. The final prediction is obtained through the function $\text{argmax}(\cdot)$.

In the above-simplified model, $\text{Attn}^{(1)}$ plays an important role in information transmission, which can be regarded as the combination of shift attention and broadcast attention.

### 5.5 Shift and broadcast in Llama2-6B

We find that the operations of shift and broadcast by attention, which are important in learning anchor functions, are also very common in LLMs. We use Llama2-6B [31] as an example to demonstrate these two operations. Similar to the identity learning task, we employ "My name is John. What is my name? Your name is" as the input for Llama2-6B and observe the attention maps at different layers. Figs. 5.5(a) and 5.5(b) illustrates two heads in Llama2-6B. The left one, a head in the second layer, exhibits noticeable shift operations, and the second one, a head in the twenty-ninth layer, demonstrates clear broadcast operations. Multiple attention maps for various heads are supplied in the appendix. Both operations are rather common.

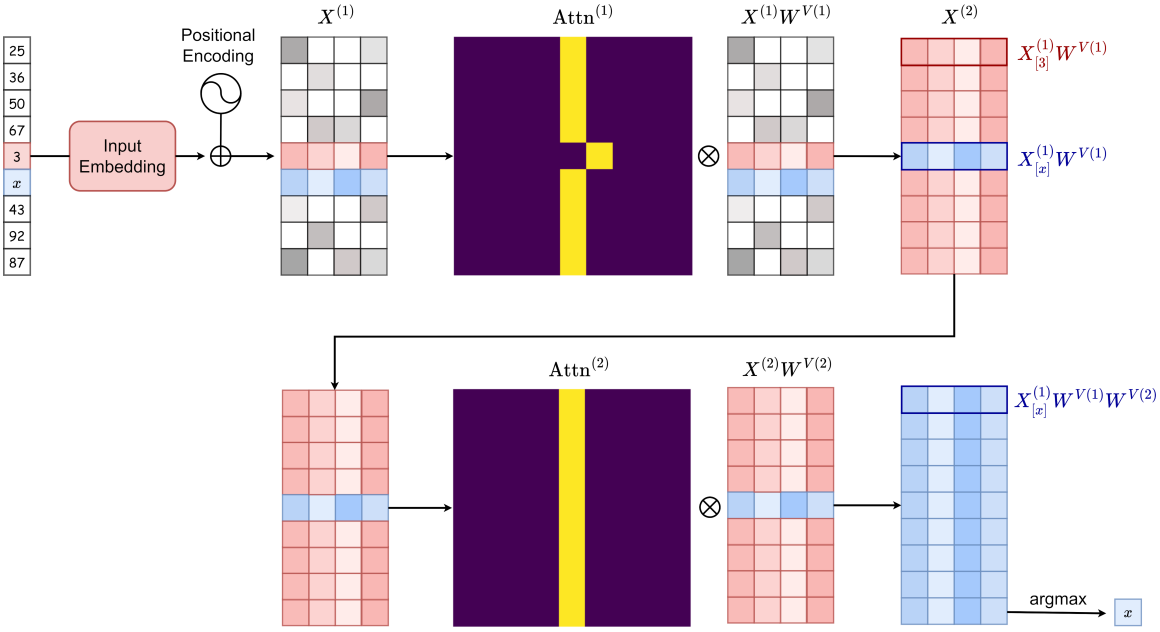
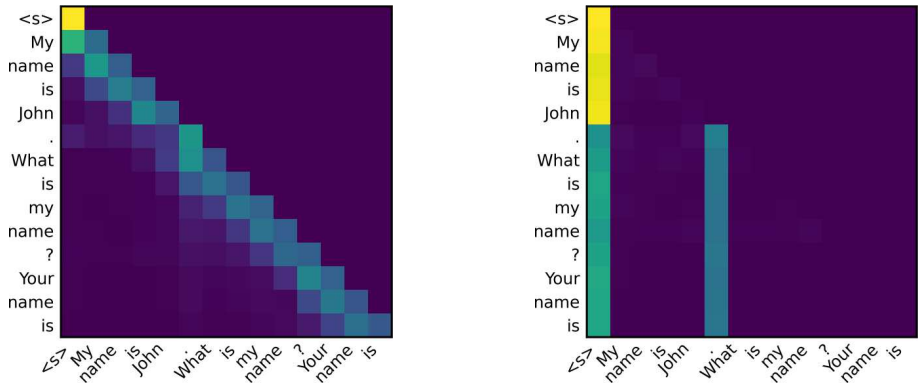



Figure 5.4: The mechanism of the 2-layer simplified transformer model for the identity learning task. The first attention matrix makes the output vector become $X^{(1)}_{[3]}W^{v(1)}$ or $X^{(1)}_{[x]}W^{v(1)}$, where $[3]$ and $[x]$ means the position of "3" and "$x$" in the input sequence. The second attention matrix makes all of the output vectors become $X^{(1)}_{[x]}W^{v(1)}W^{v(2)}$. $W^{v(1)}W^{v(2)}$ plays the role of classification.




(a) The 23-rd head of the second layer

(b) The 17-th head of the 29th layer

Figure 5.5: Attention examples of Llama2-6B.

It is noteworthy that shallow-layer attentions tend to favor shift operations, while deep-layer attentions tend to favor broadcast operations. The full-size attention map can be found in the Appendix C. Additionally, the highly similar broadcast attention patterns in deep layers explain the phenomenon where performance remains almost unaffected upon the removal of several consecutive layers that are near the final output.

### 5.6 Discussion of shift and broadcast operations

Obviously, the attention structure can perform various operations in various learning tasks. What kind of operations the attention structure shows depends on the task and also the input sequence. For the identity learning anchor function, the distance between the anchor and the key item is fixed. Then, the attention structure learns this position relation and performs a shift operation. Such a shift can always fuse the information of the anchor and the key item for any input sequence.

The broadcast operation can simplify the network, such as ignoring irrelevant tokens, providing working memory for intermediate results, making extra deep layers negligible, etc. From the perspective of complexity, broadcast operations enable neural networks to accomplish tasks using a function with low complexity.

The shift and broadcast are only two basic operations among many others. For example, the induction head correlates the current token with similar ones in the previous context [9, 24]. The future study of various anchor functions will reveal more and more basic operations of attention structure as well as other components.

## 6 Existing theoretical understanding for anchor function

In this section, we utilize some existing theoretical understanding for learning anchor functions, including frequency principle [4, 21, 26, 29, 38–40, 43] and condensation [7, 22, 46–48].

### 6.1 Frequency principle

The frequency principle indicates an implicit spectral bias during the learning process, i.e., neural networks often learn from low- to high-frequency.

Frequency is used to indicate how much change of the output is affected by the change of the input. A larger change in the output corresponds to a higher frequency. We design two tasks that have different frequencies. In each task, the anchor set $A = \{3, 4\}$, and in each input sequence, there exists only one anchor. Denote the key item following the anchor as $x$.

For task one, the output is always the item $x$. For example, the output of $(12, 33, 14, 3, 42, 22, 32, 20, 28)$ is 42, and the output of $(12, 33, 14, 4, 42, 22, 32, 20, 28)$ is also 42.

For task two, for anchor "3", the output is $x$; for anchor "4", the output is $x + 1$. For example, the output of $(12, 33, 14, 3, 42, 22, 32, 20, 28)$ is 42, and the output of $(12, 33, 14, 4, 42, 22, 32, 20, 28)$ is 43.

Obviously, task two has a higher frequency. According to the frequency principle, the learning of task two should be slower. As shown in Fig. 6.1, for both the training loss and the test accuracy, the fitting speed of the low-frequency task is faster than that of the high-frequency task.

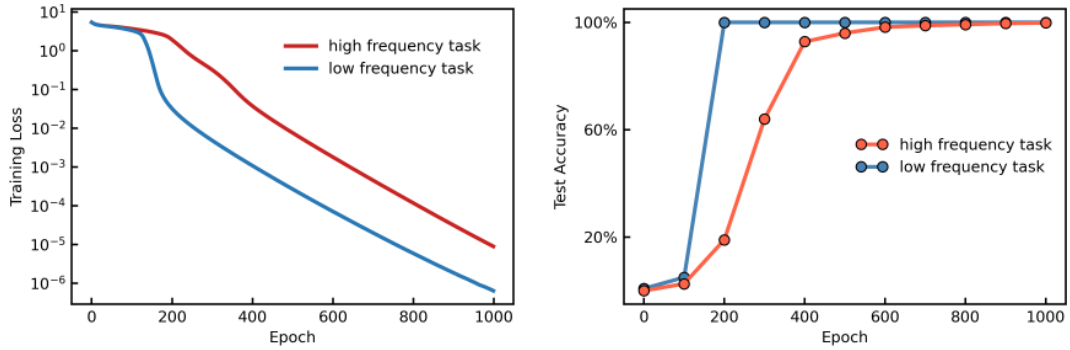



Figure 6.1: Training loss and test accuracy for two frequency tasks.

## 6.2 Condensation

Previous works have found that, as the initialization of network weights becomes smaller, the neurons in the same layer would tend to be more condensed during the training, i.e., weights of neurons in the same layer tend to be more similar. Condensation implies that neural networks tend to fit the data with a function with complexity as small as possible. A key reason underlying the condensation is that the neuron in the same layer has similar training dynamics.

For a transformer network with multiple heads in the same layer, the heads also have similar dynamics. We should expect that heads in the same layer can have many similarities during or after the training. As shown in Fig. 6.2, we show the attention map of each head in the second layer of a 4-layer 12-head model. The attention maps between different heads are highly similar. They all perform the broadcast operation.

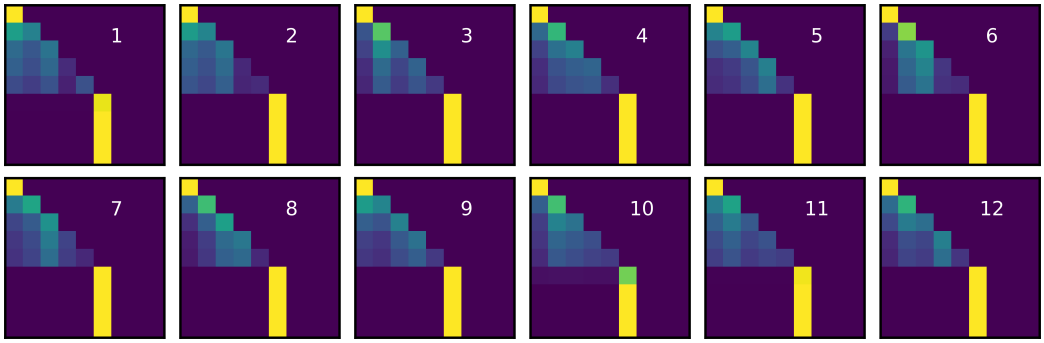



Figure 6.2: The attention map of each head in the second layer of the 4-layer 12-head model.

# 7 Future problems

The anchor function has opened up a new avenue for studying language models. The study of the anchor function in the main text, along with the detailed analysis in the ap-

pendix, has opened up a new avenue for studying language models. There are many more questions worth further exploration, such as some questions listed in Table 7.1. An exciting thing is that readers can easily cook up more language tasks with anchor functions and propose a series of interesting questions for further exploration.

The empirical discoveries enabled by the anchor function have supported a series of theoretical and mechanistic studies, aiming to explain when and why language models exhibit reasoning, memorization, and generalization. For instance, using the composite function problem in Section 4.4, recent work found that parameter initialization significantly affects whether a model favors reasoning or memorization during training [44, 45].

Table 7.1: Example questions for further exploration based on anchor function.

| Category | Question |
|---|---|
| Structure characteristic | Why transformer network is better than the fully-connected network on anchor functions? |
| | Why transformer network can generalize better than LSTM network on anchor functions with small data size? |
| | What is the effect of each structure component? such as attention, layer normalization, softmax, skip connection, and fully-connected one? |
| | Why the following can happen: When some components are removed, the generalization degrades, but when some more components are removed, the generalization improves. |
| Approximation | What is the benefit of multiple heads in approximation? |
| | What is the benefit of multiple layers in approximation? |
| Generalization & Optimization | Why transformer networks can generalize over unseen composite tasks? |
| | How does the transformer network learn synonym anchors? |
| | What is the benefit of multiple heads in training? |
| | What is the benefit of multiple layers in training? |
| | How does the landscape differ over the increasing number of heads? |
| | How does the landscape differ over the increasing number of layers? |
| | Why does the emergence of generalization over unseen composite tasks later than that over unseen data of seen tasks? |
| | Why learning rate is so sensitive? |
| | Why do loss spikes sometimes help but not always? |
| | Why do many heads learn the same attention, is it similar to the condensation phenomenon? |
| | Why forward recitation task is easier than the backward one? |
| | Is there a learning order for different layers, and why? |
| | How does the hidden space dimension affect learning? |
| | Why the pattern of the output sequence is simpler when the data size is larger? |
| | Explore more basic operations of the attention structure in various tasks. |
| More | Build more benchmark functions. |

Due to the simplicity of this data construction, researchers can conduct rigorous theoretical analyses [41,42], which have already contributed to large-scale model pretraining [11]. At the same time, studies on the Mamba architecture using this dataset have revealed learning behaviors different from those of the transformer [6]. Furthermore, through the multi-step reasoning tasks derived from Section 4.1, the generalization limits of transformers have been examined via both experiments and theory [19,25,36]. Finally, studies based on the identity learning task provide theoretical explanations for training phenomena such as regularization effects and loss spikes [3,20]. Collectively, these applications demonstrate the potential of the anchor function as a foundational framework to bridge the gap between theoretical principles and the complex behaviors of modern language models.

# Appendix A Basic settings

**Transformer.** We use a 4-layer decoder-only transformer network to learn various tasks. Each decoder layer has 4 heads. The optimizer chosen is AdamW. The batch size is set to 100. The initial learning rate is 2e-5. In the first 400 epochs, the warmup learning rate update strategy is used to gradually increase the learning rate to 2e-4, and then the cosine annealing strategy is used to decay the learning rate back to 2e-5. A total of 4000 epochs are trained.

**LSTM.** We use a 4-layer LSTM to learn various tasks. The hidden size of the model is 4. The optimizer chosen is AdamW. The batch size is set to 10. The initial learning rate is 2e-5. In the first 400 epochs, the warmup learning rate update strategy is used to gradually increase the learning rate to 2e-4, and then the cosine annealing strategy is used to decay the learning rate back to 2e-5. A total of 4000 epochs are trained.

**FNN.** We use a 4-layer FNN to learn various tasks. The hidden size of the model is 4. The optimizer chosen is AdamW. The batch size is set to 10. The initial learning rate is 2e-5. In the first 400 epochs, the warmup learning rate update strategy is used to gradually increase the learning rate to 2e-4, and then the cosine annealing strategy is used to decay the learning rate back to 2e-5. A total of 4000 epochs are trained.

# Appendix B Two-layer transformer

In this section, we study the mechanism of two-layer transformers to achieve the identity task. We first give a detailed notation of the transformer structure, and then select three modules that have a greater contribution to this task, namely the first-layer attention module, the first-layer feedforward neural network (FFN) module and the second-layer attention module to analyze its mechanism, shown in Fig. B.1.

## B.1 Transformer architecture details

In this section, we give the notation of the transformer in modules to facilitate subsequent analysis of its mechanism.

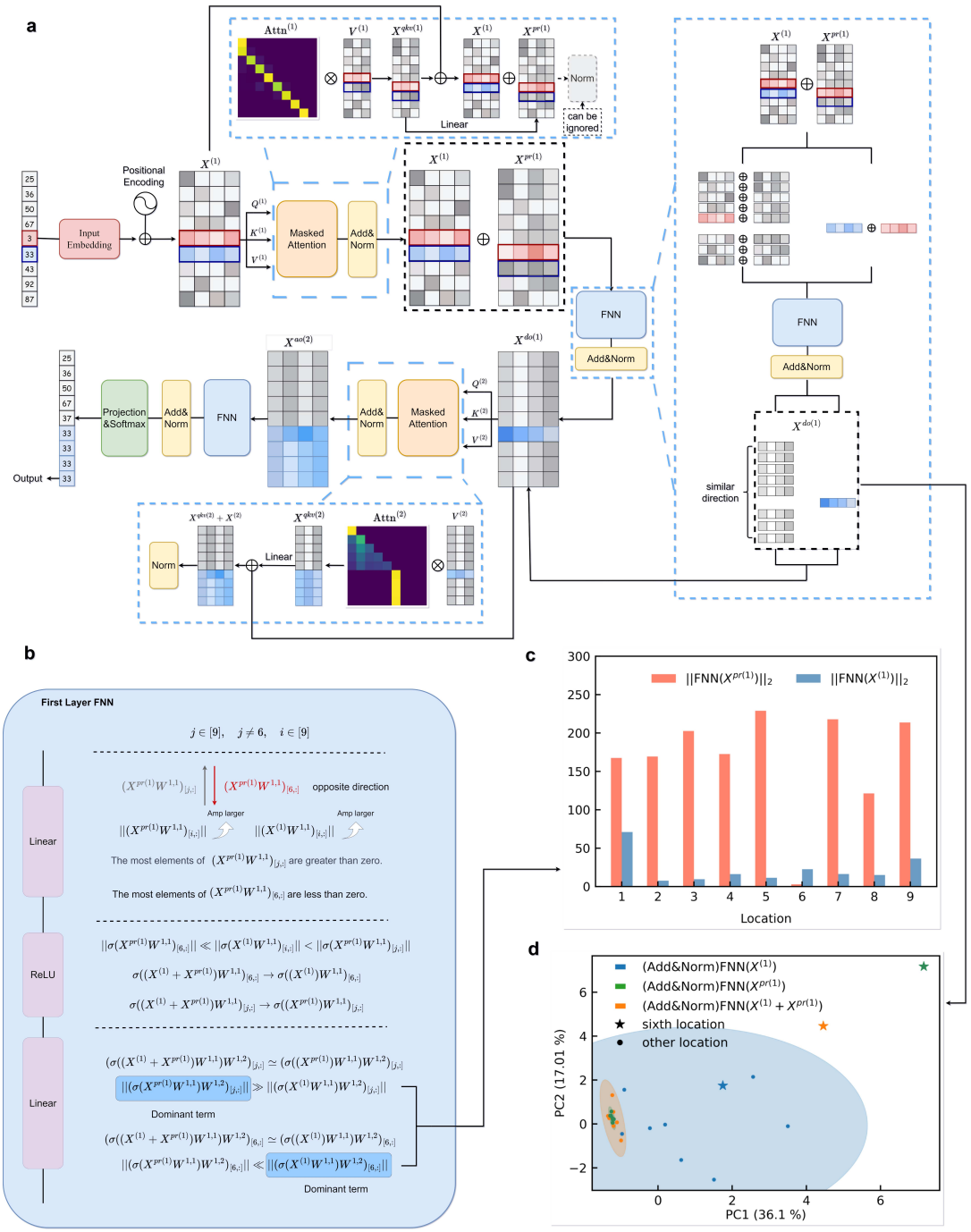



Figure B.1: Detailed mechanism of a two-layer transformer. (a) Flow chart of two-layer transformer. (b) The detailed internal mechanism of the first layer FFN. (c) The norm of $\mathrm{FFN}(X^{(1)})$ and $\mathrm{FFN}(X^{\mathrm{pr}(1)})$ with respect to different tokens. (d) PCA Analysis of three kinds of FFN outputs, i.e., (Add&Norm)FFN$(X^{(1)})$, (Add&Norm)FFN$(X^{\mathrm{pr}(1)})$ and (Add&Norm)FFN$(X^{(1)}+X^{\mathrm{pr}(1)})$. "(Add&Norm)" indicates that output undergoes residual connection and layer normalization.

### B.1.1 Input representation

The input sequence is represented as a one-hot vector $X^{\text{in}} \in \mathbb{R}^{n\times d}$, where $n$ is the sequence length, $d$ is the dictionary size, and $X^{\text{in}}$ is the one-hot vector.

After embedding, the input sequence becomes

$$X^{\text{em}} = X^{\text{in}} W^{\text{em}} \in \mathbb{R}^{n\times d_m}, \quad W^{\text{em}} \in \mathbb{R}^{d\times d_m}.$$

The positional vector is denoted as

$$X^{\text{pos}} \in \mathbb{R}^{n\times d_m}.$$

The input to the first transformer block is

$$X^{(1)} = X^{\text{em}} + X^{\text{pos}}.$$

### B.1.2 Attention mechanism

For the $l$-th layer, the $Q, K, V$ matrices of the attention mechanism are defined as functions of the input $X^{(l)} \in \mathbb{R}^{n\times d_m}$

$$\begin{aligned}
Q^{(l)}(X^{(l)}) &= X^{(l)} W^{Q(l)}, \quad W^{Q(l)} \in \mathbb{R}^{d_m\times d_k},\\
K^{(l)}(X^{(l)}) &= X^{(l)} W^{K(l)}, \quad W^{K(l)} \in \mathbb{R}^{d_m\times d_k},\\
V^{(l)}(X^{(l)}) &= X^{(l)} W^{V(l)}, \quad W^{V(l)} \in \mathbb{R}^{d_m\times d_k}.
\end{aligned}$$

The attention matrix $\text{Attn}^{(l)}(X^{(l)})$ for the $l$-th layer is computed as

$$\text{Attn}^{(l)}(X^{(l)}) = \text{softmax}\left(\frac{Q^{(l)}(X^{(l)}) K^{(l)}(X^{(l)})^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{n\times n},$$

where the function $\text{softmax}(\cdot)$ is defined as

$$\text{Softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}.$$

The output of the attention mechanism, denoted as $X^{\text{qkv}(l)}$, is given by

$$X^{\text{qkv}(l)} = \text{Attn}^{(l)}(X^{(l)}) V^{(l)}(X^{(l)}) \in \mathbb{R}^{n\times d_k}.$$

Finally, the output after position-wise feedforward processing is obtained using residual connection and layer normalization

$$X^{\text{pr}(l)} = X^{\text{qkv}(l)} W^{\text{attn},l}, \quad W^{\text{attn},l} \in \mathbb{R}^{d_k\times d_m}, \quad X^{\text{pr}(l)} \in \mathbb{R}^{n\times d_m}.$$

The output of the $l$-th layer, $X^{\text{ao}(l)}$, is computed as

$$X^{\text{ao}(l)} = \text{LayerNorm}(X^{(l)} + X^{\text{pr}(l)}).$$

### B.1.3 Feedforward neural network

The $l$-th layer feedforward neural network is expressed as

$$X^{\mathrm{do}(l)} := \mathrm{FFN}(X^{\mathrm{ao}(l)}) = \mathrm{LayerNorm}\big(X^{\mathrm{ao}(l)} + \sigma(X^{\mathrm{ao}(l)}W^{l,1})W^{l,2}\big),$$

where $\sigma(\cdot) = \mathrm{ReLU}(\cdot)$, and $W^{l,1} \in \mathbb{R}^{d_m \times 3d_m}$, $W^{l,2} \in \mathbb{R}^{3d_m \times d_m}$.

### B.1.4 Projection layer

The projection layer is defined as

$$Y = X^{\mathrm{do}(L)}W^{\mathrm{proj}} + b^{\mathrm{proj}}, \quad W^{\mathrm{proj}} \in \mathbb{R}^{d_m \times d}, \quad b^{\mathrm{proj}} \in \mathbb{R}^{d}, \quad Y \in \mathbb{R}^{n \times d},$$

where $n = 9, d_m = 400, L = 2, d_k = 64$. The output is obtained by taking the argmax of the softmax

$$\mathrm{Output} = \mathrm{argmax}\big(\mathrm{softmax}(Y)\big).$$

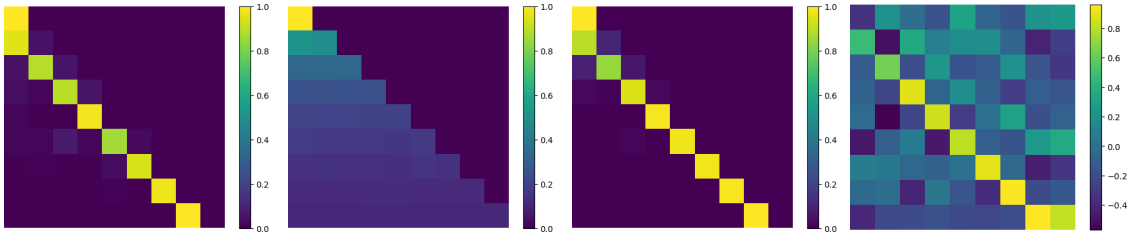
Here, the anchor "3" is positioned at the 5-th position, and the target position "$x$" is at the 6-th position in the sequence. For clarity, vector positions in the sequence may be referenced in subsequent discussions.

## B.2 Analysis of attention matrix $\mathrm{Attn}^{(1)}$

The structure of the attention matrix $\mathrm{Attn}^{(1)}$ is mainly determined by the positional encoding vector $X^{\mathrm{pos}}$ shown in Figs. B.2(a), B.2(b) and B.2(c). To gain insights into the structure of the attention matrix, we examine the cosine similarity between $Q^{(1)}(X^{\mathrm{pos}})$ and $K^{(1)}(X^{\mathrm{pos}})$, as shown in the following Fig. B.2(d), and find that the direction of $Q^{(1)}(X^{\mathrm{pos}})_i$ and $K^{(1)}(X^{\mathrm{pos}})_{i+1}$ is almost the same. It is noteworthy that in the first layer's attention mechanism, the prediction results remain correct if we artificially replace the attention matrix with

$$\mathrm{Attn}^{(1)}(X^{(1)}) = \begin{bmatrix} 0 & & & & & & \\ 1 & 0 & & & & & \\ 0 & 1 & 0 & & & & \\ \vdots & \vdots & \vdots & & & & \\ 0 & 0 & 0 & \dots & 1 & 0 & \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \end{bmatrix}.$$

This suggests that the fine structure of attention in the first layer does not significantly affect the outcomes. Additionally, the layer normalization following the attention mechanism does not impact the results. Therefore, the main purpose of $\mathrm{Attn}^{(1)}$ is to complete the shift of the vector and combine it with the subsequent residual connection to fuse the key item information with the anchor information.



(a) $\text{Attn}^{(1)}(X^{(1)})$ (b) $\text{Attn}^{(1)}(X^{em})$ (c) $\text{Attn}^{(1)}(X^{\text{pos}})$ (d) Cosine similarity

Figure B.2: (a, b, c) Attention output of the first layer with different input $X^{(1)}, X^{em}$ and $X^{\text{pos}}$. (d) The cosine similarity between $Q^{(1)}(X^{\text{pos}})$ and $K^{(1)}(X^{\text{pos}})$.

## B.3 Analysis of feedforward neural network

The FFN tends to align the vectors excluding the sixth position, i.e., $X_i^{\text{do}(1)}, i \neq 6$, in parallel while making the vector at the sixth position, $X_6^{\text{do}(1)}$, as orthogonal as possible to the other vectors.

This alignment and orthogonality are primarily influenced by the output $X^{\text{pr}(1)}$ from the first layer's attention mechanism. In other words, the ResNet operation applied to $X^{(1)}$ in the first layer of FFN alters the direction of the input vector $X^{(1)}$ to some extent, resulting in pairwise parallelism and orthogonality in $X^{\text{do}(1)}$. It is worth noting that this similarity and orthogonality are not immediately apparent in $X^{\text{ao}(l)}$, even though $X^{\text{ao}(l)}$ already incorporates information from $X^{\text{pr}(1)}$. These properties become more pronounced after passing through the FFN.

The following is the experimentally verification of these observations. Experimental results suggest that the layer normalization of the first layer's attention has minimal impact on the results, thus we assume for convenience that

$$X^{\text{ao}(1)} := \text{LayerNorm}(X^{(1)} + X^{\text{pr}(1)}) = X^{(1)} + X^{\text{pr}(1)}.$$

We examine the cosine similarity between vectors at different positions and stages of the FFN. The FFN is divided into six stages: Input (the FFN's input vector), First linear output, ReLU output, Second linear output, ResNet output, and Layernorm output.

As shown in Fig. B.3, the FFN tends to align the vectors (excluding the sixth position) and orthogonalize the vector at the sixth position. And the parallelism of these vectors is also reflected in $\text{FFN}(X^{(1)} + X^{\text{pr}(1)})$ (see Fig. B.4). This phenomenon can also be further characterized using PCA. As shown in Fig. B.5, asterisks represent quantities corresponding to the sixth position digit, and circles represent quantities at other positions. The close alignment of the blue asterisk and circles suggests minimal distributional differences. Conversely, the orange asterisk, influenced by the addition of $X^{\text{pr}(1)}$, causes the circles to become relatively concentrated (closer to the green circles), while the asterisk moves farther away (closer to the green asterisk).

The conversion of vector directions through the FFN layer is mainly to facilitate the construction of the second layer attention matrix. In the next subsection, we delve deeper into the generation of the second-layer attention matrix.

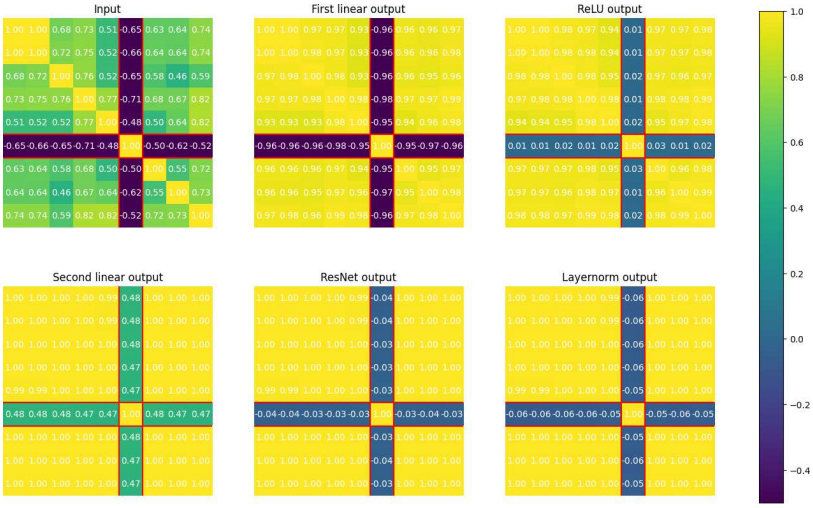



Figure B.3: Cosine similarity analysis for the output in different first-layer FFN stages with input $X^{\mathrm{pr}(1)}$. Each subfigure represents in turn: the input vector, the output of the first linear layer, the output after the activation function, the output of the second linear layer, the output after residual connection, and the output after layer normalization.

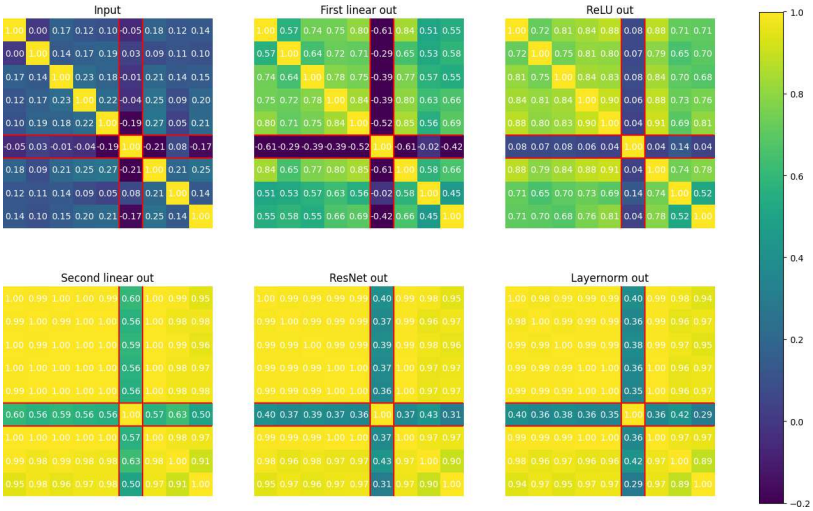



Figure B.4: Cosine similarity analysis for the output in different first-layer FFN stages with input $X^{(1)} + X^{\mathrm{pr}(1)}$. Each subfigure represents in turn: the input vector, the output of the first linear layer, the output after the activation function, the output of the second linear layer, the output after residual connection, and the output after layer normalization.

## B.4 Analysis of attention matrix $\mathrm{Attn}^{(2)}$

Fig. B.6 shows the $\mathrm{Attn}^{(2)}$ matrix before and after mask and softmax operations. We only need to figure out the cause of the attention matrix before these two operations.

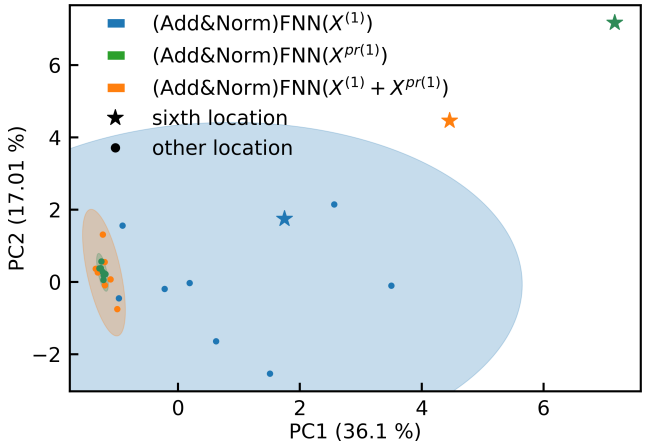
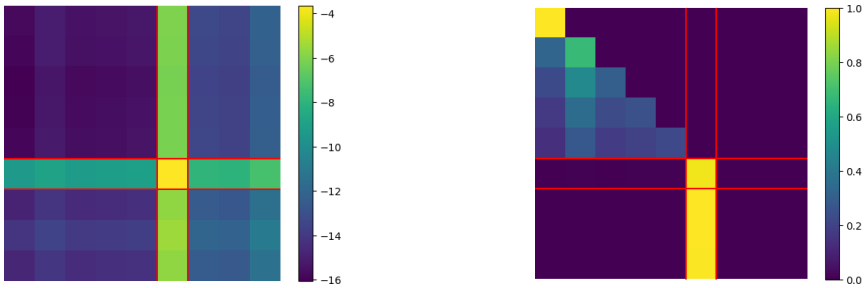



Figure B.5: PCA analysis of three kinds of FFN outputs, i.e., (Add&Norm)FFN($X^{(1)}$), (Add&Norm)FFN($X^{\text{pr}(1)}$) and (Add&Norm)FFN($X^{(1)} + X^{\text{pr}(1)}$). "(Add&Norm)" indicates that output undergoes residual connection and layer normalization. Asterisks represent quantities corresponding to the sixth position digit, while circles represent quantities at other positions. The blue asterisk and circles are almost coincident, indicating minimal distributional differences. The orange asterisk, influenced by the addition of $X^{\text{pr}(1)}$, causes the circles to become relatively concentrated (closer to the green circles), while the asterisk moves farther away (closer to the green asterisk).



(a) (b)

Figure B.6: Analysis of the second-layer attention mechanism before and after masking and softmax. (a) The mechanism before applying masking and softmax. (b) The mechanism with broadcast functionality after masking and softmax.

We mentioned earlier that after FFN, the row vector of $X^{(2)}$ has only 2 directions. We denote the vector of the position outside the key item as $\alpha_0$, and the vector of the position of the key item as $\beta_0$. Therefore, as shown in Fig. B.7, the row vectors of the matrix $Q^{(2)}$ and $K^{(2)}$ in the second layer also have approximately only two directions. And the two directions of matrix $Q^{(2)}$ and $K^{(2)}$ have a roughly opposite relationship. Therefore, matrix $Q^{(2)}$ and $K^{(2)}$ can be approximately expressed as

$$Q^{(2)} = (\alpha^T, \ldots, \alpha^T, \beta^T, \alpha^T, \ldots, \alpha^T)^T,$$
$$K^{(2)} = (-\alpha^T, \ldots, -\alpha^T, -\beta^T, -\alpha^T, \ldots, -\alpha^T)^T.$$

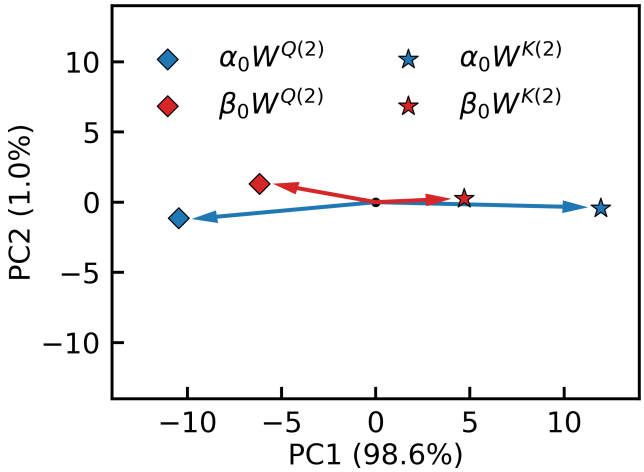



Figure B.7: The PCA of the first and sixth row vector of $Q$ and $K$ in the second layer. $\alpha_0 W^{Q(2)}, \beta_0 W^{Q(2)}$, $\alpha_0 W^{K(2)}$, and $\beta_0 W^{K(2)}$ represent the first raw vector of $Q^{(2)}$, the sixth raw vector of $Q^{(2)}$, the first raw vector of $K^{(2)}$, and the sixth raw vector of $K^{(2)}$, respectively.

Therefore, we have

$$Q^{(2)}K^{(2)T} = -\begin{pmatrix} \alpha\alpha^T & \dots & \alpha\alpha^T & \alpha\beta^T & \alpha\alpha^T & \dots & \alpha\alpha^T \\ \vdots & \ddots & \vdots & \alpha\beta^T & \vdots & \ddots & \vdots \\ \alpha\alpha^T & \dots & \alpha\alpha^T & \alpha\beta^T & \alpha\alpha^T & \dots & \alpha\alpha^T \\ \beta\alpha^T & \dots & \beta\alpha^T & \beta\beta^T & \beta\alpha^T & \dots & \beta\alpha^T \\ \alpha\alpha^T & \dots & \alpha\alpha^T & \alpha\beta^T & \alpha\alpha^T & \dots & \alpha\alpha^T \\ \vdots & \ddots & \vdots & \alpha\beta^T & \vdots & \ddots & \vdots \\ \alpha\alpha^T & \dots & \alpha\alpha^T & \alpha\beta^T & \alpha\alpha^T & \dots & \alpha\alpha^T \end{pmatrix}. \tag{B.1}$$

To realize the matrix in Fig. B.6(a), just need $\alpha\beta^T > 0$ and $\|\alpha\|_2 > \|\beta\|_2$.

## B.5 Interpretation of attention mechanism results

After the second layer of the decoder block, the last row vector of hidden state $X^{(3)}$ is

$$X_9^{(3)} = \text{Layernorm}\big(\alpha_0 + \beta_0 W^{V(2)}\big),$$

where $\alpha_0$ is the vector of positions other than the position of the key item and $\beta_0$ is the vector of the key item position. $\alpha_0$ has no information about the coupling of anchor and key item, so its response to the projection layer should not have an impact on the final result. As can be seen from Fig. B.8, $\alpha_0 W^{\text{proj}} + b^{\text{proj}}$ makes the probability of outputting 20-100 greater than other tokens, but the probability of outputting 20-100 is approximately the same.

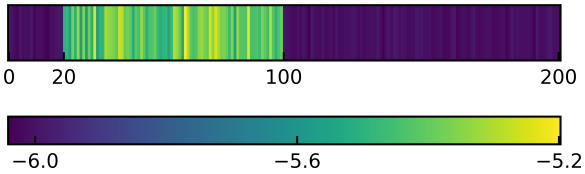



Figure B.8: Projection output of $\alpha_0$ passing through the projection layer. $\alpha_0$ makes the model tend to output numbers in the range 20-100. But it has little impact on the specific output value.

# Appendix C Attention maps of Llama2-6B

To better understand how the model implements the proposed computation, we visualize the attention matrices of Llama2-6B on our synthetic tasks. Fig. C.1 reports attention maps across several representative layers and heads (ordered from top to bottom and left to right). This visualization highlights an intuitive transition of attention patterns: early layers tend to exhibit a "shift"-like behavior that mainly propagates local positional infor-

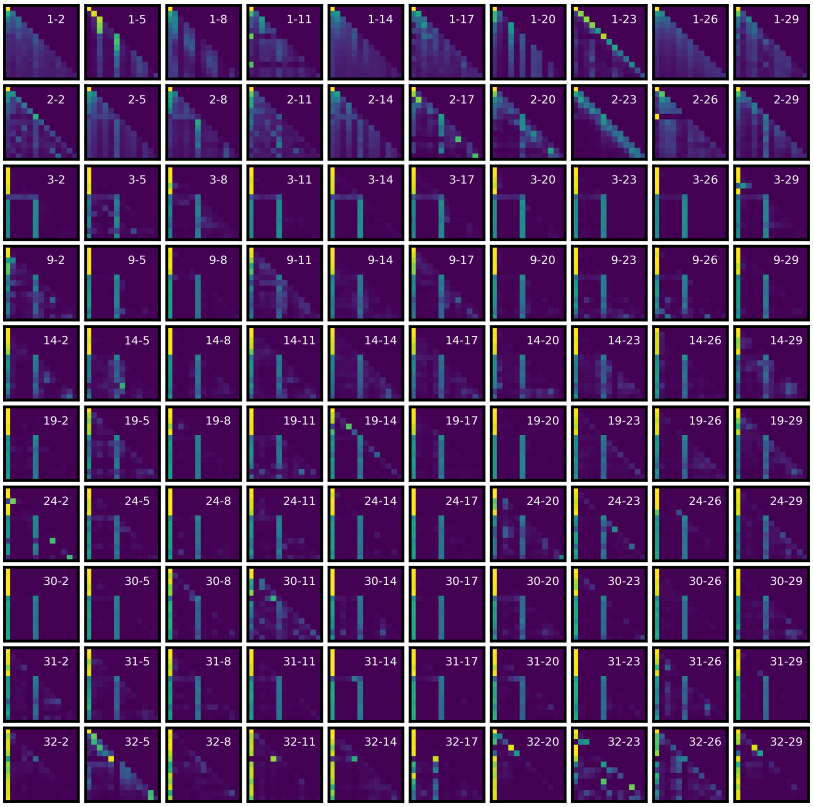
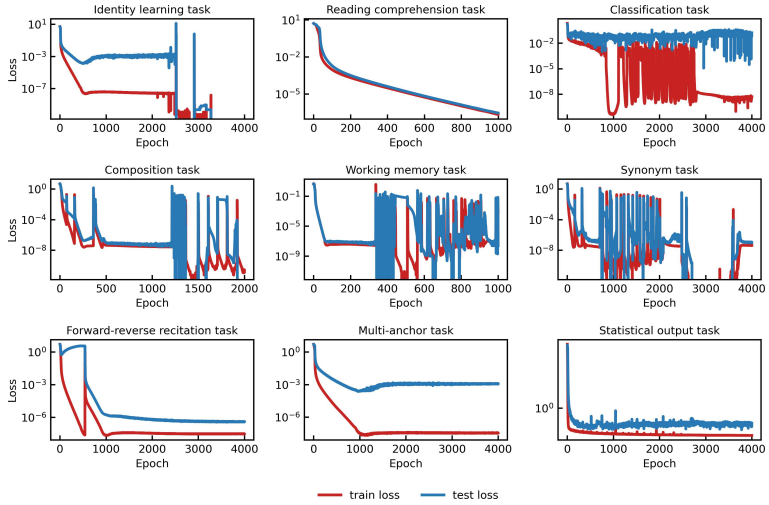



Figure C.1: Attention maps of Llama2-6B. From top to bottom, from left to right are the attention matrices of different layers and heads. In the first few layers, attention mostly shows the pattern of shift. In the later layers, attention mostly shows the pattern of the broadcast.

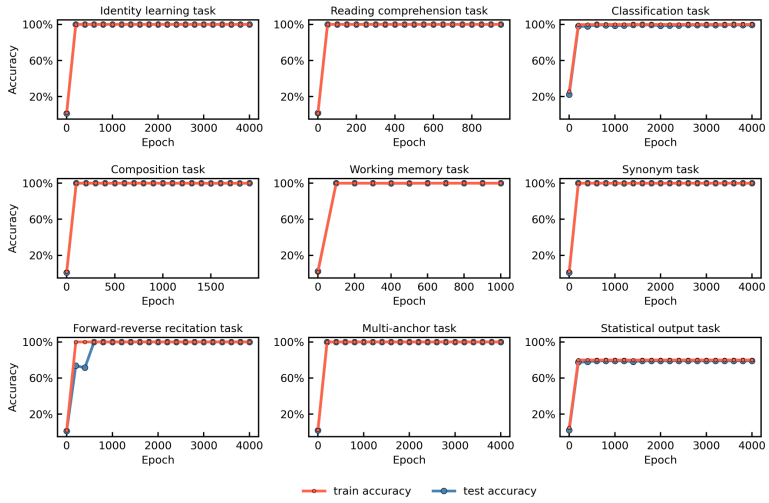
mation, whereas deeper layers increasingly show a "broadcast"-like behavior that aggregates and distributes information globally. These qualitative patterns are consistent with our mechanistic interpretation that different layers specialize in different sub-routines of the overall algorithm.

# Appendix D Curve of accuracy and loss

In this section, we report the loss curves (Fig. D.1(a)) and accuracy curves (Fig. D.1(b)) of the transformer model when learning each task.




(a) Loss




(b) Accuracy

Figure D.1: Loss and accuracy of each task.

## Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant Nos. 92270001, 12371511, 12422119), and by the National Key R&D Program of China (Grant No. 2022YFA1008200).

## References

[1] K. Ahn, X. Cheng, H. Daneshmand, and S. Sra, Transformers learn to implement preconditioned gradient descent for in-context learning, *arXiv:2306.00297*, 2023.

[2] E. Akyürek, D. Schuurmans, J. Andreas, T. Ma, and D. Zhou, What learning algorithm is in-context learning? Investigations with linear models, *arXiv:2211.15661*, 2022.

[3] Z. Bai, Z. Zhou, J. Zhao, X. Li, Z. Li, F. Xiong, H. Yang, Y. Zhang, and Z.-Q. J. Xu, Adaptive preconditioners trigger loss spikes in Adam, *arXiv:2506.04805*, 2025.

[4] Y. Cao, Z. Fang, Y. Wu, D.-X. Zhou, and Q. Gu, Towards understanding the spectral bias of deep learning, in: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 2205–2211, 2021.

[5] S. Chan, A. Santoro, A. Lampinen, J. Wang, A. Singh, P. Richemond, J. McClelland, and F. Hill, Data distributional properties drive emergent in-context learning in transformers, in: *Advances in Neural Information Processing Systems*, Vol. 35, Curran Associates, Inc., 18878–18891, 2022.

[6] T. Chen, P. Lin, Z. Wang, and Z.-Q. J. Xu, Achilles' Heel of Mamba: Essential difficulties of the Mamba architecture demonstrated by synthetic data, *arXiv:2509.17514*, 2025.

[7] Z. Chen, Y. Li, T. Luo, Z. Zhou, and Z.-Q. J. Xu, Phase diagram of initial condensation for two-layer neural networks, *CSIAM Trans. Appl. Math.*, 5(3):448–514, 2024.

[8] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, What does BERT look at? An analysis of BERT's attention, *arXiv:1906.04341*, 2019.

[9] N. Elhage et al., Framework for investigating transformer circuits, `https://transformer-circuits.pub/2021/framework/index.html`, 2021.

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

[11] L. Hang et al., Scalable complexity control facilitates reasoning ability of LLMs, *arXiv:2505.23013*, 2025.

[12] G. E. Hinton, To recognize shapes, first learn to generate images, in: *Progress in Brain Research*, Vol. 165, Elsevier, 535–547, 2007.

[13] H. Jiang and Q. Li, Approximation theory of transformer networks for sequence modeling, *arXiv:2305.18475*, 2023.

[14] J. Kaplan et al., Scaling laws for neural language models, *arXiv:2001.08361*, 2020.

[15] O. Kovaleva, A. Romanov, A. Rogers, and A. Rumshisky, Revealing the dark secrets of BERT, *arXiv:1908.08593*, 2019.

[16] B. M. Lake and M. Baroni, Human-like systematic generalization through a meta-learning neural network, *Nature*, 623:115–121, 2023.

[17] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, The omniglot challenge: A 3-year progress report, *Curr. Opin. Behav. Sci.*, 29:97–104, 2019.

[18] Z. Li, Y. Cao, C. Gao, Y. He, H. Liu, J. Klusowski, J. Fan, and M. Wang, One-layer transformer provably learns one-nearest neighbor in context, in: *Advances in Neural Information Processing Systems*, Vol. 37, Curran Associates, Inc., 82166–82204, 2024.

[19] P. Lin, Z.-A. Chen, and Z.-Q. J. Xu, Identity bridge: Enabling implicit reasoning via shared latent memory, *arXiv:2509.24653*, 2025.

[20] P. Lin, Z. Zhang, and Z.-Q. J. Xu, Reasoning bias of next token prediction training, *arXiv:2502.02007*, 2025.

[21] T. Luo, Z. Ma, Z.-Q. J. Xu, and Y. Zhang, Theory of the frequency principle for general deep neural networks, *CSIAM Trans. Appl. Math.*, 2(3):484–507, 2021.
[22] T. Luo, Z.-Q. J. Xu, Z. Ma, and Y. Zhang, Phase diagram for two-layer relu neural networks at infinite-width limit, *J. Mach. Learn. Res.*, 22(71):1–47, 2021.
[23] C. Ma and L. Ying, Why self-attention is natural for sequence-to-sequence problems? A perspective from symmetries, *arXiv:2210.06741*, 2022.
[24] C. Olsson et al., In-context learning and induction heads, *arXiv:2209.11895*, 2022.
[25] T. Qin, Y. Chen, Z. Wang, and Z.-Q. J. Xu, Limit analysis for symbolic multi-step reasoning tasks with information propagation rules based on transformers, *arXiv:2509.23178*, 2025.
[26] N. Rahaman, D. Arpit, A. Baratin, F. Draxler, M. Lin, F. A. Hamprecht, Y. Bengio, and A. Courville, On the spectral bias of deep neural networks, in: *Proceedings of the 36th International Conference on Machine Learning*, PMLR, 97:5301–5310, 2019.
[27] G. Reddy, The mechanistic basis of data dependence and abrupt learning in an in-context classification task, *arXiv:2312.03002*, 2023.
[28] R. Rende, F. Gerace, A. Laio, and S. Goldt, Mapping of attention mechanisms to a generalized Potts model, *Phys. Rev. Res.*, 6:023057, 2024.
[29] B. Ronen, D. Jacobs, Y. Kasten, and S. Kritchman, The convergence rate of neural networks for learned functions of different frequencies, in: *Advances in Neural Information Processing Systems*, Vol. 32, Curran Associates, Inc., 4761–4771, 2019.
[30] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, Meta-learning with memory-augmented neural networks, in: *Proceedings of The 33rd International Conference on Machine Learning*, PMLR, 48:1842–1850, 2016.
[31] H. Touvron et al., Llama 2: Open foundation and fine-tuned chat models, *arXiv:2307.09288*, 2023.
[32] L. Van der Maaten and G. Hinton, Visualizing data using t-SNE, *J. Mach. Learn. Res.*, 9(86):2579–2605, 2008.
[33] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov, Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned, *arXiv:1905.09418*, 2019.
[34] J. Von Oswald, E. Niklasson, E. Randazzo, J. Sacramento, A. Mordvintsev, A. Zhmoginov, and M. Vladymyrov, Transformers learn in-context by gradient descent, in: *Proceedings of the 40th International Conference on Machine Learning*, PMLR, 202:35151-35174, 2023.
[35] L. Wang, L. Li, D. Dai, D. Chen, H. Zhou, F. Meng, J. Zhou, and X. Sun, Label words are anchors: An information flow perspective for understanding in-context learning, *arXiv:2305.14160*, 2023.
[36] Z. Wang, Y. Wang, Z. Zhang, Z. Zhou, H. Jin, T. Hu, J. Sun, Z. Li, Y. Zhang, and Z.-Q. J. Xu, Understanding the language model to solve the symbolic multi-step reasoning problem from the perspective of buffer mechanism, in: *Findings of the Association for Computational Linguistics: EMNLP 2025*, Association for Computational Linguistics, 16446–16474, 2025.
[37] Z. Wang, S. Wei, D. Hsu, and J. D. Lee, Transformers provably learn sparse token selection while fully-connected nets cannot, *arXiv:2406.06893*, 2024.
[38] Z.-Q. J. Xu, Y. Zhang, and T. Luo, Overview frequency principle/spectral bias in deep learning, *arXiv:2201.07395*, 2022.
[39] Z.-Q. J. Xu, Y. Zhang, T. Luo, Y. Xiao, and Z. Ma, Frequency principle: Fourier analysis sheds light on deep neural networks, *Commun. Comput. Phys.*, 28(5):1746–1767, 2020.
[40] Z.-Q. J. Xu, Y. Zhang, and Y. Xiao, Training behavior of deep neural network in frequency domain, in: *Neural Information Processing. ICONIP 2019. Lecture Notes in Computer Science*, Vol. 11953, Springer, 264–274, 2019.
[41] J. Yao, Z. Zhang, and Z.-Q. J. Xu, An analysis for reasoning bias of language models with small initialization, in: *Proceedings of the 42nd International Conference on Machine Learning*, PMLR, 267:71834–718642025, 2025.
[42] J. Yao and Z.-Q. J. Xu, Probability signature: Bridging data semantics and embedding structure in language models, *arXiv:2509.20124*, 2025.
[43] Y. Zhang, T. Luo, Z. Ma, and Z.-Q. J. Xu, A linear frequency principle model to understand the absence of overfitting in neural networks, *Chin. Phys. Lett.*, 38(3):038701, 2021.

[44] Z. Zhang, P. Lin, Z. Wang, Y. Zhang, and Z.-Q. Xu, Initialization is critical to whether transformers fit composite functions by reasoning or memorizing, in: *Advances in Neural Information Processing Systems*, Vol. 37, Curran Associates, Inc., 14093–14126, 2024.

[45] Z. Zhang, P. Lin, Z. Wang, Y. Zhang, and Z.-Q. J. Xu, Complexity control facilitates reasoning-based compositional generalization in transformers, *IEEE Trans. Pattern Anal. Mach. Intell.*, 2025, doi: 10.1109/TPAMI.2025.3646483.

[46] H. Zhou, Z. Qixuan, T. Luo, Y. Zhang, and Z.-Q. Xu, Towards understanding the condensation of neural networks at initial training, in: *Advances in Neural Information Processing Systems*, Vol. 35, Curran Associates, Inc., 2184–2196, 2022.

[47] H. Zhou, Q. Zhou, Z. Jin, T. Luo, Y. Zhang, and Z.-Q. J. Xu, Empirical phase diagram for three-layer neural networks with infinite width, in: *Advances in Neural Information Processing Systems*, Vol. 35, Curran Associates, Inc., 26021– 26033, 2022.

[48] Z. Zhou, H. Zhou, Y. Li, and Z.-Q. J. Xu, Understanding the initial condensation of convolutional neural networks, *CSIAMTrans. Appl. Math.*, 6(2):272–319, 2025.