Interpreting Arithmetic Reasoning in Large Language Models using Game-Theoretic Interactions

Anonymous ACL submission

Abstract

In recent years, large language models (LLMs) have made significant advancements in arithmetic reasoning. However, the internal mechanism of how LLMs solve arithmetic problems remains unclear. In this paper, we propose explaining arithmetic reasoning in LLMs using game-theoretic interactions. Specifically, we disentangle the output score of the LLM into numerous interactions between the input words. We quantify different types of interactions encoded by LLMs during forward propagation to explore the internal mechanism of LLMs for solving arithmetic problems. We find that (1) the internal mechanism of LLMs for solving simple one-operator arithmetic problems is their capability to encode operand-operator interaction patterns and high-order interaction patterns from input samples. Additionally, we find that LLMs with poor arithmetic capabilities focus more on context-free interactions. (2) The internal mechanism of LLMs for solving relatively complex two-operator arithmetic problems is their capability to encode operator interaction patterns from input samples. (3) An LLM gradually forgets its capability to solve simple one-operator arithmetic problems as it learns to solve relatively complex two-operator arithmetic problems[1].

1 Introduction

In recent years, the arithmetic reasoning capabilities of large language models (LLMs) have improved significantly, but the internal mechanism is still unclear. Some studies identified neurons that have great effects on arithmetic reasoning (Yu and Ananiadou, 2024; Rai and Yao, 2024). Other studies evaluated the impact of each word/token in the input arithmetic question on neuron activations and network outputs (Stolfo et al., 2023; Zhang et al., 2024). However, previous studies have not mathematically guaranteed that the explanations faithfully reflect the arithmetic reasoning logic of LLMs. Besides, a DNN does not independently use each word/token for inference. Rather, a DNN relies on interactions among input variables for inference (Ren et al., 2023a). For example, the interaction between "*green*" and "*hand*" forms the concept of "*beginner*." To this end, a series of recent studies have used game-theoretic interactions to explain the representation power of traditional DNNs (Ren et al., 2021a,b; Deng et al., 2021), which have been proven to be faithful explanations by a series of theoretical guarantees (Ren et al., 2023a).

Inspired by these studies, we use interactions to explain the arithmetic reasoning capability of LLMs. As Figure 1 shows, given an input arithmetic question with n words, e.g., "*What is 2 plus 7? Answer is*," the interaction S (e.g., $S = \{2, plus, 7\}$) represents an AND relationship between the words in S, which is **equivalently**[2] encoded by the LLM. Each interaction S contributes a numerical effect I_S to push the output score towards generating the answer "*9*." It has been proven that **the output score of a DNN on any masked[3] input sample can be represented as the sum of the effects of all triggered interactions (Ren et al., 2023a), which guarantees that the interaction truly reflects the inference logic of the DNN.** Please see Theorem 1 for details.

In this way, interactions can be considered as inference patterns encoded by an LLM. Therefore, we quantify how LLMs encode different interactions during forward propagation. We find that different LLMs have preferences for encoding certain interactions. For example, as Figure 2 shows, when computing features closer to the output layer, the

[1] All code and data will be released upon acceptance.

[2] Note that each interaction is equivalently encoded by the entire DNN, rather than by a specific neuron.

[3] It is common to use a specific token or embedding to mask the input variables of a DNN, but there are still no unified masking strategies. Please see Appendix A for an introduction to masking strategies.

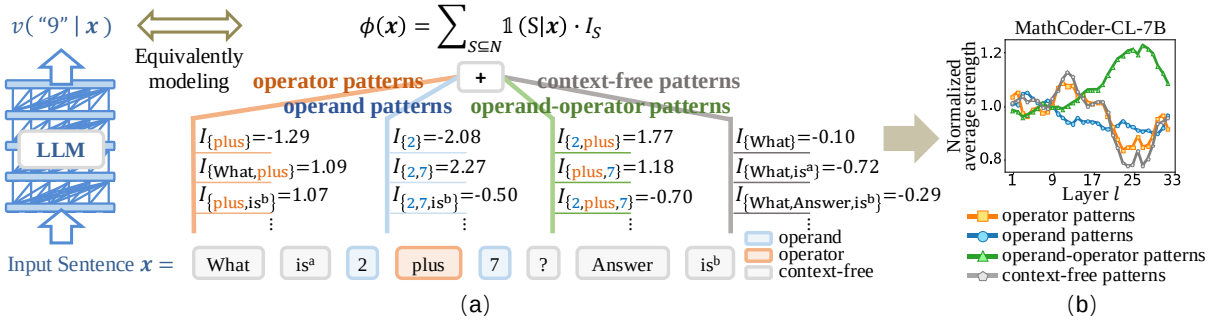

Figure 1: (a) Illustration of using interactions to explain the outputs of LLMs for arithmetic problems. Given an arithmetic question $\boldsymbol{x}$, let N be a set containing all the words in $\boldsymbol{x}$. We use a logic model based on interactions, i.e., $\phi(\boldsymbol{x}) = \sum_{S\subseteq N} \mathbb{1}(S \mid \boldsymbol{x}) \cdot I_S$ to explain the detailed inference patterns encoded by LLMs. Note that superscripts a and b are not input to the LLM, but are used to distinguish two words of "*is*" in two positions. (b) Based on interactions, we explore how an LLM encodes different types of interaction patterns during forward propagation.

Llemma-7B (Azerbayev et al., 2023) model tends to encode interactions that contain both operands and operators, while the OPT-1.3B (Zhang et al., 2022) model tends to encode interactions that only contain context-free words. Inspired by the above observations, we further define four types of interaction patterns encoded by the LLM for arithmetic reasoning, including (1) operand interaction patterns, (2) operator interaction patterns, (3) operand-operator interaction patterns, and (4) context-free interaction patterns, as shown in Figure 1. Furthermore, we propose a new metric to quantify the focus of LLMs on the different types of interaction patterns above and discover the following insights.

• *Insight 1 (Internal mechanism of solving simple one-operator problems)*: **The internal mechanism of LLMs for solving simple one-operator arithmetic problems is their capability to encode[2] operand-operator interaction patterns and high-order interaction patterns from input samples.** For simple one-operator arithmetic problems, LLMs that perform well tend to enhance the strength of operand-operator interaction patterns and high-order interaction patterns during forward propagation. A high-order interaction contains a large number of input words, while a low-order interaction contains a small number of input words. In contrast, LLMs that perform poorly tend to focus mostly on context-free interaction patterns and extremely low-order interaction patterns.

• *Insight 2 (Internal mechanism of solving relatively complex two-operator problems)*: **The internal mechanism of LLMs for solving relatively complex two-operator arithmetic problems is their capability to encode[2] operator interaction patterns from input samples.** The LLM tends to focus more on operator interaction patterns throughout the process of learning to solve two-operator arithmetic problems.

• *Insight 3*: As an LLM learns to solve relatively complex two-operator problems, its capability to solve simple one-operator problems declines. Meanwhile, the LLM tends to reduce its focus on operand-operator interaction patterns and high-order interaction patterns.

2 Related Work

Explaining arithmetic reasoning in LLMs. Some studies identified and analyzed key neurons to explain arithmetic reasoning in LLMs. Yu and Ananiadou (2024) identified an internal logic chain. Rai and Yao (2024) investigated neuron activations and identified reasoning-related neurons. Some studies attempted further analysis by changing the inputs of the LLM. Zhang et al. (2024) identified key components by perturbing activations and measuring the changes in the output logits. Stolfo et al. (2023) examined the impact of changes in each input unit on neuron activations and network outputs. However, previous studies failed to provide strong mathematical support for their explanations. In contrast, in this paper, we use interactions to explain how LLMs solve arithmetic tasks, which has been proven to be a faithful explanation.

Using game-theoretic interactions to explain DNNs. Ren et al. (2023a) proposed using interactions to explain DNNs and provided a series of theoretical guarantees to ensure the faithfulness of this explanation. A series of studies further explored using interactions to explain the representational power of DNNs, including adversarial robust-

ness (Wang et al., 2021), adversarial transferability (Wang et al., 2020), and the overfitting problem (Ren et al., 2023b; Zhou et al., 2023). However, due to terabytes of data and billions of parameters in LLMs, as well as the inherent complexity of arithmetic reasoning tasks, whether interactions can be used to explain the arithmetic reasoning of LLMs while ensuring the faithfulness of the explanation remains to be verified.

3 Using interactions to explain inference patterns encoded by LLMs

3.1 Preliminaries: Disentangling the network output using interactions

Given a DNN and an input sample $\boldsymbol{x}$ with n variables indexed by $N = \{1, 2, \ldots, n\}$, we can obtain 2^n masked[3] sentences $\{\boldsymbol{x}_T \mid T \subseteq N\}$ by masking[3] each of n variables. Specifically, $\boldsymbol{x}_T$ denotes the input sentence when we keep variables in T unchanged and mask[3] variables in $N \backslash T$. $\boldsymbol{x}_\emptyset$ denotes the input sentence when all variables in $\boldsymbol{x}$ are masked[3]. Ren et al. (2021b) have proven that there exists a surrogate logical model $\phi(\cdot)$ that can well predict/fit the scalar output of the DNN for all 2^n different masked sentences, as Theorem 1 shows.

$$\forall T \subseteq N, \phi(\boldsymbol{x}_T) = \phi(\boldsymbol{x}_\emptyset) + \sum_{S \subseteq N, S \neq \emptyset} \mathbb{1}(S|\boldsymbol{x}_T) \cdot I_S, \quad (1)$$

the trigger function $\mathbb{1}(S|\boldsymbol{x}_T)$ represents an **AND** relationship among the input variables in a set $S \subseteq N$. That is, if all variables in S are present in T, then $\mathbb{1}(S|\boldsymbol{x}_T) = 1$. Otherwise, if any variable in S is masked[3], then $\mathbb{1}(S|\boldsymbol{x}_T) = 0$. Each interaction S has a scalar weight I_S.

Theorem 1. *(Universal matching property, as proven by Ren et al. (2023a)). When the scalar weight I_S in the logical model $\phi(\cdot)$ is set as follows,*

$$\forall S \subseteq N, S \neq \emptyset, I_S = \sum_{S' \subseteq S} (-1)^{|S|-|S'|} \cdot v(\boldsymbol{x}_{S'}), \quad (2)$$

then we have $\forall T \subseteq N$, the output score of the DNN $v(\boldsymbol{x}_T) = \phi(\boldsymbol{x}_T)$.

The universal matching property guarantees the theoretical faithfulness of using interactions to explain the DNN. That is, the interaction truly reflects the inference logic of the DNN. Besides, the interaction has been proven to serve as the basis for many game-theoretic metrics, further ensuring its theoretical faithfulness. Please see Appendix C for more details about game-theoretic metrics.

3.2 Explaining the LLM using interactions

Although interactions have been widely used to explain traditional DNNs, the following two challenges remain in using interactions to explain the forward propagation process of the LLM.

(1) How to quantify interactions encoded by an LLM in intermediate layers. Given an LLM with L layers[4], we follow previous work (Wendler et al., 2024) by focusing on the embedding of the last input token at each layer l. We conduct experiments, detailed in Appendix E, showing that the mid-layer features of other tokens remain nearly unchanged when the input is masked[3]. At layer l, we set the output score $v^{(l)}(\boldsymbol{x}_T)$ as follows.

$$v^{(l)}(\boldsymbol{x}_T) = \frac{(f^{(l)}(\boldsymbol{x}_N))^\top \cdot f^{(l)}(\boldsymbol{x}_T)}{\|f^{(l)}(\boldsymbol{x}_N)\|_2 \cdot \|f^{(l)}(\boldsymbol{x}_T)\|_2}, \quad (3)$$

where $f^{(l)} \in \mathbb{R}^d$ denotes the embedding of the last input token and $\|\cdot\|_2$ is the L2-norm. The score $v^{(l)}(\boldsymbol{x}_T)$ measures the utility of $f^{(l)}(\boldsymbol{x}_T)$ along the direction of $f^{(l)}(\boldsymbol{x}_N)$. Since significant masking typically removes feature strength along the feature direction $\frac{f^{(l)}(\boldsymbol{x}_{N\backslash T})}{\|f^{(l)}(\boldsymbol{x}_{N\backslash T})\|_2}$, changes in the score $v^{(l)}(\boldsymbol{x}_T)$ for differently masked sentences $\boldsymbol{x}_T$ are attributed to interaction effects on the l-th layer's feature. Thus, the interactions computed based on the scores $v^{(l)}(\boldsymbol{x}_T)$ for all $T \subseteq N$ can be considered as being encoded by features at the l-th layer. Note that we use $f^{(l)}(\boldsymbol{x}_T) - f^{(l)}(\boldsymbol{x}_\emptyset)$ to replace $f^{(l)}(\boldsymbol{x}_T)$, where $f^{(l)}(\boldsymbol{x}_\emptyset)$ denotes the embedding when we mask[3] all input variables in $\boldsymbol{x}$. This shifting operation is used to ensure that $f^{(l)}(\boldsymbol{x}_\emptyset) = \mathbf{0}$.

(2) We use words instead of tokens as input variables. It is because the tokenizers of different LLMs adopt different strategies for encoding numbers. For example, the Llama-2-7B (Touvron et al., 2023) model divides the word "*15*" into two tokens "*1*" and "*5*," while the OPT-1.3B (Zhang et al., 2022) model obtains a single token "*15*." To ensure a fair comparison of interactions encoded by different LLMs, we treat all tokens within a word as a single input variable. When generating the masked[3] sentence $\boldsymbol{x}_T$, we use a specific padding token (see Appendix B for details) to mask all tokens that belong to the words in $N \setminus T$.

Base on the above two settings, we quantify interactions in LLMs when solving arithmetic problems. Figure 2 shows the top 10 interactions (i.e., those interactions with largest absolute values) encoded[2]

[4]Please see Appendix D for details about network architectures and chosen layers.

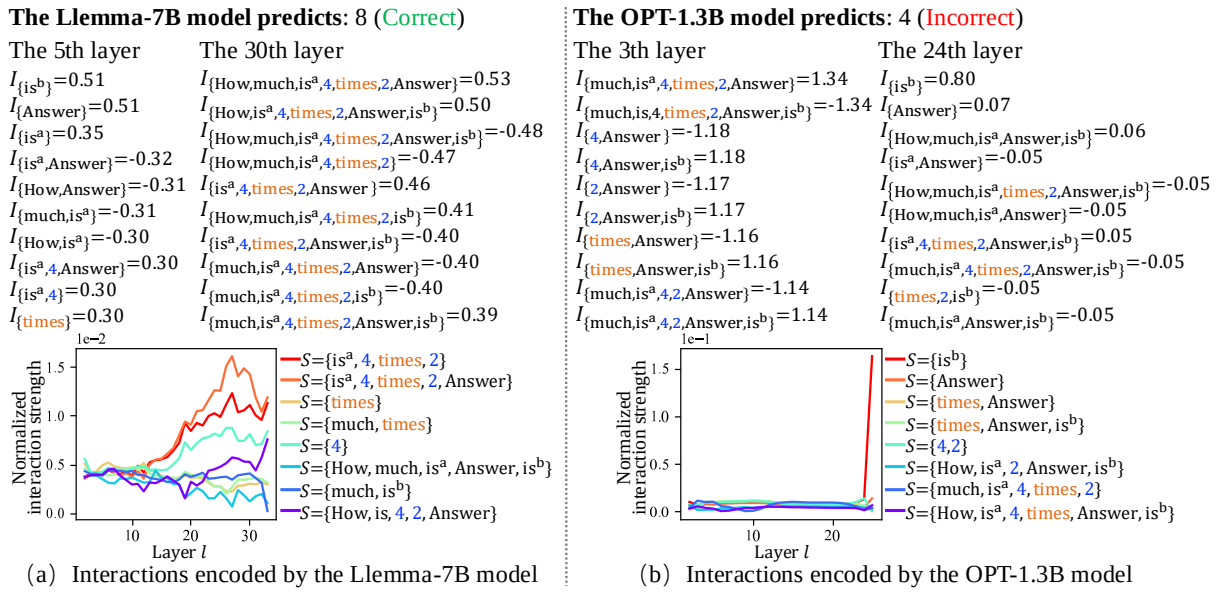

Figure 2: Visualization of top 10 interactions encoded by LLMs when computing features in different layers. Results show that the Llemma-7B model mainly focuses on interactions containing both operands (in blue) and operators (in orange) in a late layer (i.e., the 30th layer). In comparison, the OPT-1.3B model mainly focuses on interactions that contain only context-free words in a late layer (i.e., the 24th layer), which may be the reason why the OPT-1.3B model answers incorrectly. Note that superscripts a and b are not input to the LLM, but are used to distinguish two words of "*is*" in two positions.

by the Llemma-7B model (Azerbayev et al., 2023) and the OPT-1.3B model (Zhang et al., 2022) when solving an one-operator problem "*How much is 4 times 2? Answer is*." The Llemma-7B model predicts the correct answer "*8*," while the OPT-1.3B model predicts the incorrect answer "*4*." We observe that the Llemma-7B model mainly focuses on interactions that contain only context-free words in an early (5th) layer[5], and mainly focuses on interactions containing both operands and operators in a late (30th) layer. In comparison, the OPT-1.3B model mainly focuses on interactions that contain only context-free words in a late (24th) layer, which might be the reason why the OPT-1.3B model answers incorrectly.

To better observe the changing trends of different interactions encoded by an LLM during forward propagation, Figure 2 reports the curves of interactions encoded by LLMs when computing features. To ensure a fair comparison of interactions at different layers, we compute the normalized strength of interactions at different layers, that is, $\frac{|I_S^{(l)}|}{Z^{(l)}}$, where $Z^{(l)} = \mathbb{E}_{S\subseteq N}|I_S^{(l)}|$. As Figure 2 shows, the Llemma-7B model enhances its focus on interactions containing both operands and operators (e.g., $S = \{is^a, 4, times, 2\}$) during forward propagation. In comparison, the OPT-1.3B model does not show a preference for any interactions in the middle layers and shifts its focus to interactions that contain only context-free words (e.g., $S = \{is^b\}$) in the very few layers close to the output layer. Therefore, interactions provide us with a new perspective to explain the internal mechanism of LLMs for arithmetic reasoning.

[5] Please see Appendix D for details about network architectures and chosen layers.

3.3 Defining and quantifying different types of interaction patterns

Inspired by the above observations obtained in Section 3.2, we classify the variables in an input arithmetic query $\boldsymbol{x}$ into the following three categories: operand variables $\boldsymbol{x}^{\text{opd}}$, operator variables $\boldsymbol{x}^{\text{opr}}$ and context-free variables $\boldsymbol{x}^{\text{ctx-free}}$. Thus, all 2^n interactions can be classified as the following four types to describe the inference patterns encoded by LLMs for arithmetic reasoning.

• *Operand interaction patterns.* If an interaction S contains at least one operand variable but no operator variables, we regard S as an operand pattern. Let Ω^{opd} denote the set of all operand patterns.

$$\Omega^{\text{opd}} = \{S \mid \exists\, \boldsymbol{x}^{\text{opd}} \in S \wedge \nexists\, \boldsymbol{x}^{\text{opr}} \in S\}. \quad (4)$$

• *Operator interaction patterns.* If an interaction S contains at least one operator variable but no operand variables, we regard S as an operator pattern. Let Ω^{opr} denote the set of all operator patterns.

$$\Omega^{\text{opr}} = \{S \mid \exists\, \boldsymbol{x}^{\text{opr}} \in S \wedge \nexists\, \boldsymbol{x}^{\text{opd}} \in S\}. \tag{5}$$

• *Operand-operator interaction patterns.* If an interaction S contains at least one operand variable and at least one operator variable, we regard S as an operand-operator pattern. Let $\Omega^{\text{opd-opr}}$ denote the set of all operand-operator patterns.

$$\Omega^{\text{opd-opr}} = \{S \mid \exists\, \boldsymbol{x}^{\text{opd}} \in S \wedge \exists\, \boldsymbol{x}^{\text{opr}} \in S\}. \tag{6}$$

• *Context-free interaction patterns.* If an interaction S contains neither operand nor operator variables, we regard S as a context-free pattern. Let $\Omega^{\text{ctx-free}}$ denote the set of all context-free patterns.

$$\Omega^{\text{ctx-free}} = \{S \mid \nexists\, \boldsymbol{x}^{\text{opd}} \in S \wedge \nexists\, \boldsymbol{x}^{\text{opr}} \in S\}. \tag{7}$$

Note that the union of the four sets above is the complete set of 2^n interactions. **According to Theorem 1, the sum of numerical effects of these four types of interactions can accurately fit the output scores of the LLM, thereby ensuring the faithfulness of the analysis.** We design the following metric to quantify how an LLM focuses on a specific type of interaction $\Omega^{type} \in \{\Omega^{\text{opd}}, \Omega^{\text{opr}}, \Omega^{\text{opd-opr}}, \Omega^{\text{ctx-free}}\}$.

Definition 1. *(Focality on a specific type of interaction). The focality on a specific type of interaction Ω^{type} at layer l, $R^{(l)}(\Omega^{type})$, is computed as follows,*

$$R^{(l)}(\Omega^{type}) = \frac{\mathbb{E}_{S\in\Omega^{type}}|I_S^{(l)}|}{Z^{(l)}}, \tag{8}$$

where $Z^{(l)} = \mathbb{E}_{S\subseteq N}|I_S^{(l)}|$ is a normalization term used to ensure a fair comparison of the interaction effects across different layers.

In Equation 8, a higher $R^{(l)}(\Omega^{type})$ value suggests that the LLM focuses more on this type of interaction Ω^{type} when computing features at layer l. If $R^{(l)}(\Omega^{type}) = 1$, the strength of this type of interaction Ω^{type} encoded by the LLM is equal to the average strength of all interactions encoded by the LLM, which reveals that the LLM does not show a preference for this type of interaction.

We also quantify interactions of different orders to measure the representation complexity of an LLM. The order m of an interaction S is defined as the number of variables in S, that is, $m = |S|$. We design the following metric to quantify how an LLM focuses on m-order interactions.

Model	1-opr	2-opr
OPT-1.3B	6.7	5.3
GPT-J-6B	28.0	9.1
Llama-2-7B	77.7	17.8
CodeLlama-2-7B	75.0	13.8
MathCoder-L-7B	78.0	12.6
MathCoder-CL-7B	70.7	12.7
Llemma-7B	88.1	22.2
OPT-1.3B Fine-tuned	97.0	69.9

Table 1: Overall accuracy (%) of different LLMs on one-operator and two-operator arithmetic queries.

Definition 2. *(Focality on interactions of a specific order). The focality on m-order interactions at layer l, $\kappa_m^{(l)}$, is computed as follows,*

$$\forall m \in \{1, 2, \ldots, n\}, \kappa_m^{(l)} = \frac{\mathbb{E}_{|S|=m}|I_S^{(l)}|}{Z^{(l)}}. \tag{9}$$

If $\kappa_m^{(l)}$ has a larger value when m is higher, it indicates that the LLM encodes interactions of greater complexity when computing features at layer l. If $\kappa_m^{(l)}$ has a larger value when m is lower, it indicates that the LLM encodes interactions of lower complexity when computing features at layer l.

4 Comparative studies

In this section, we conduct comparative studies to analyze the internal mechanism of LLMs for arithmetic reasoning (see Section 4.1). We also fine-tune an LLM to improve its capability to solve arithmetic problems and explore how the LLM encodes different interaction patterns during the training process (see Section 4.2).

LLMs. We use interactions to analyze seven LLMs for arithmetic reasoning, including the OPT-1.3B (Zhang et al., 2022) model, the GPT-J-6B (Wang and Komatsuzaki, 2021) model, the Llama-2-7B (Touvron et al., 2023) model, the CodeLlama-2-7B (Roziere et al., 2023) model, the MathCoder-L-7B (Wang et al., 2023) model, the MathCoder-CL-7B (Wang et al., 2023) model, and the Llemma-7B (Azerbayev et al., 2023) model. Appendix B shows how to mask words for these LLMs.

Data. We follow Karpas et al. (2022); Razeghi et al. (2022); Stolfo et al. (2023) to conduct experiments on a set of arithmetic problems handcrafted by humans, including 6 templates for one-operator two-operand queries and 29 templates for two-operator three-operand queries. For example, "*The sum of n_1 and n_2 is*" and "*What is the ratio between n_1 minus n_2 and n_3? The answer is.*" Each template for one-operator queries includes all four arithmetic operators, i.e., $\{+, -, \times, \div\}$, and each

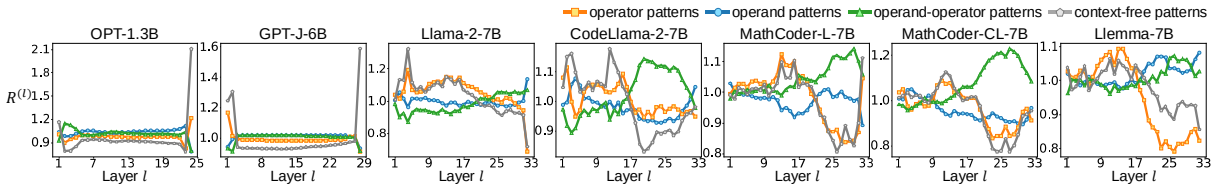

Figure 3: Comparing the normalized average strength $R^{(l)}$ of different types of interaction patterns encoded by LLMs during forward propagation. Each curve in the figure is averaged over various one-operator arithmetic queries.

template for two-operator queries corresponds to a unique combination of two operators. Please see Appendix F for template details. For each template of one-operator queries, we generate 20 prompts by randomly sampling operands (n_1, n_2)[6], and for each template of two-operator queries, we generate 5 prompts following the same procedure.

Table 1 shows the overall accuracy of different LLMs on one-operator queries and two-operator queries. We observe that the Llama-2-7B model, the CodeLlama-2-7B model, the MathCoder-L-7B model, the MathCoder-CL-7B model, and the Llemma-7B model perform well on one-operator queries, while the OPT-1.3B model and the GPT-J-6B model perform relatively poorly. However, for two-operator queries, all seven LLMs perform poorly. Please see Appendix G for accuracy test results on each template.

4.1 Exploring the internal mechanism of LLMs for solving arithmetic problems

In this subsection, we analyze the focality on different interaction patterns of LLMs during forward propagation and obtain the following insights.

Insight 1: The internal mechanism of LLMs for solving simple one-operator arithmetic problems is their capability to encode operand-operator interaction patterns and high-order interaction patterns. Figure 3 reports the focality on different types of interaction patterns $R^{(l)}$ encoded by LLMs during forward propagation. The results are averaged over all one-operator queries. We observe that for simple one-operator problems, in the Llama-2-7B model, the CodeLlama-2-7B model, the MathCoder-L-7B model, the MathCoder-CL-7B model, and the Llemma-7B model, $R^{(l)}(\Omega^{\text{opd-opr}})$ gradually increases starting from the middle layers and takes the lead in the late layers. This suggests that LLMs with good arithmetic capabilities tend to use more operand-operator interactions for arithmetic reasoning during the later stages of forward propagation. Although these LLMs gradually compress the encoding of operand-operator interaction patterns as they approach the output layer ($R^{(l)}(\Omega^{\text{opd-opr}})$ of very few layers close to the output layer tends to decrease), the encoding of these interaction patterns in the late layers is sufficient to support their arithmetic reasoning. In comparison, in the OPT-1.3B model and the GPT-J-6B model, $R^{(l)}$ remains around 1.0 in most layers, while $R^{(l)}(\Omega^{\text{ctx-free}})$ suddenly increases in the very few layers close to the output layer. This suggests that LLMs with poor arithmetic capabilities do not exhibit a preference for any interaction patterns in the middle layers and focus excessively on context-free interaction patterns in the very few layers close to the output layer.

Note that, We also observe that in Figure 3, among all models, the MathCoder-CL-7B model shows the highest focus on operand-operator interaction patterns in the late layers. However, its accuracy on one-operator queries is not the highest (see Table 1), which may be because it overly limits the encoding of both operator interaction patterns and operand interaction patterns in the late layers.

Figure 4 reports the focality on interaction patterns of different orders $\kappa^{(l)}$ encoded by LLMs during forward propagation. The results are averaged over queries from a single one-operator template, as different templates correspond to different maximum orders, i.e., the number of input words. We observe that for simple one-operand problems, in the Llama-2-7B model, the CodeLlama-2-7B model, the MathCoder-L-7B model, the MathCoder-CL-7B model, and the Llemma-7B model, $\kappa_m^{(l)}$ for $m \in \{6, 7, 8\}$ tends to increase staring from the middle layers. This suggests that LLMs with good arithmetic capabilities tend to use more high-order interactions for arithmetic reasoning during the later stage of forward propagation. In comparison, in the OPT-1.3B model and the GPT-J-6B model, most $\kappa^{(l)}$ in the middle layers is around 1.0, while in the very few layers close to the

[6]We sample operands from $\{1, 2, \ldots, 9\}$ since some LLMs tokenize each digit as an independent token, such as the Llama-2-7B model.

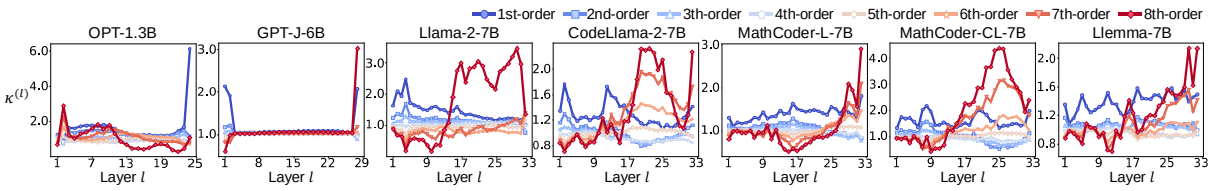

Figure 4: Comparing the normalized average strength $\kappa^{(l)}$ of interaction patterns of different orders encoded by LLMs during forward propagation. Each curve in the figure is averaged over various one-operator arithmetic queries.

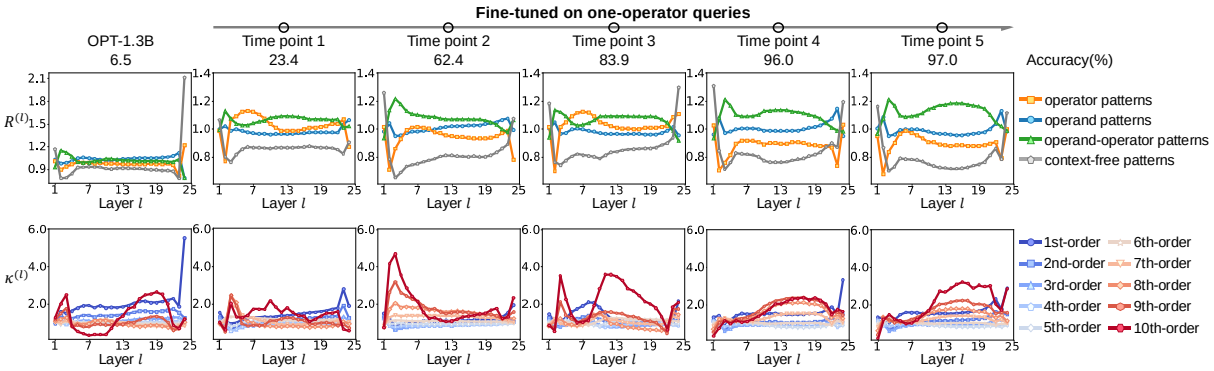

Figure 5: Visualizing the dynamic changes of different interaction patterns encoded by the OPT-1.3B model during the training process on one-operator queries. Each curve is averaged over various one-operator queries. Results show that as accuracy increases, the OPT-1.3B model tends to enhance the strength of operand-operator interaction patterns (in the upper part) and the strength of high-order interaction patterns (in the lower part).

output layer, $\kappa_1^{(l)}$ suddenly increases. This suggests that LLMs with poor arithmetic capabilities tend to excessively focus on extremely low-order interaction patterns. More results from other templates in Appendix I lead to the same conclusion.

4.2 Dynamic encoding of different interaction patterns during training process

We further explore how an LLM learns to solve arithmetic problems. That is, we investigate how an LLM encodes different interaction patterns when trained on arithmetic problem data. To this end, we fine-tune the OPT-1.3B model on arithmetic queries in the following three different ways[7]. (1) We fine-tune the OPT-1.3B model on one-operator queries, increasing its accuracy on one-operator queries from 6.7% to 97.0%. This version is termed the *OPT-1.3B-One* model. (2) We fine-tune the OPT-1.3B model on two-operator queries, increasing its accuracy on two-operator queries from 5.3% to 69.9%. (3) Building upon the *OPT-1.3B-one* model, we further train it on two-operator queries. We analyze the dynamic encoding of different interaction patterns during the above three training processes and obtain the following insights.

Figure 5 reports the dynamic changes of different interaction patterns encoded by the OPT-1.3B model during training on one-operator queries. The results of $R^{(l)}$ are averaged over all one-operator queries, and the results of $\kappa^{(l)}$ are averaged over queries from a single one-operator template[8]. We observe that as the accuracy of the OPT-1.3B model on one-operator queries increases, $R^{(l)}(\Omega^{\text{opd-opr}})$ and $\kappa_m^{(l)}$ for $m \in \{8, 9, 10\}$ gradually increase in the middle layers. **The results validate our Insight 1.**

Note that, we also observe that the OPT-1.3B model consistently enhances its focus on context-free interaction patterns in the very few layers close to the output layer during the training process. This may be due to the inherent preference of the OPT-1.3B model, which tends to enhance its focus on context-free interaction patterns in the very few layers close to the output layer, as Figure 3 shows.

Insight 2: The internal mechanism of LLMs for solving relatively complex two-operator arithmetic problems is their capability to encode operator interaction patterns. Figure 6 reports the dynamic changes of different interaction patterns encoded by the OPT-1.3B model during training on two-operator queries. The results are

[7] Please see Appendix H for details about model training.

[8] As different templates correspond to different maximum orders, please see Appendix I for more results from other templates, which lead to the same conclusion.

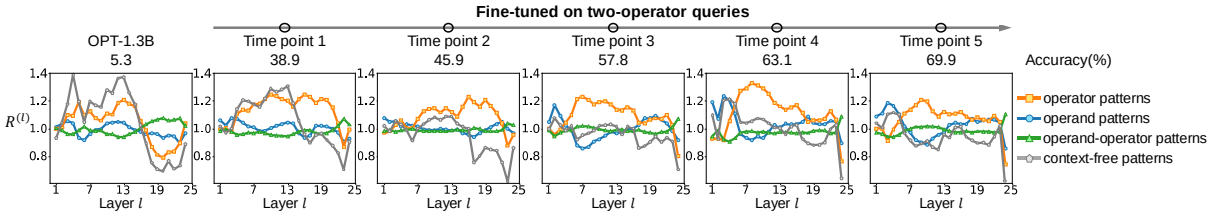

Figure 6: Visualizing the dynamic changes of different types of interaction patterns encoded by the OPT-1.3B model during the training process on two-operator queries. Each curve is averaged over various two-operator queries. Results show that the OPT-1.3B model tends to consistently focus more on operator interaction patterns.

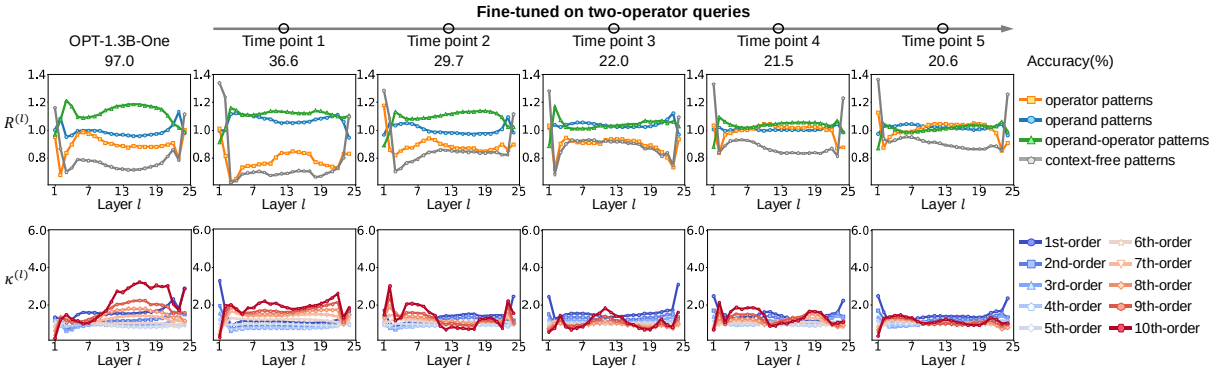

Figure 7: Visualizing the dynamic changes of different interaction patterns encoded by the *OPT-1.3B-One* model during the training process on two-operator queries. Each curve is averaged over various one-operator queries. Results show that as accuracy decreases, the *OPT-1.3B-One* model tends to decrease the strength of operand-operator interaction patterns (in the upper part) and the strength of high-order interaction patterns (in the lower part).

averaged over all two-operator queries. We observe that as the accuracy of the OPT-1.3B model on two-operator queries increases, $R^{(l)}(\Omega^{\text{opr}})$ remains consistently high in the middle layers. This suggests that solving relatively complex two-operator arithmetic problems requires an LLM to focus more on operator interaction patterns.

Insight 3: When an LLM that can solve simple one-operator arithmetic problems learns to solve relatively complex two-operator arithmetic problems, the LLM progressively loses its capability to solve the simpler ones. Figure 7 reports the dynamic changes of different interaction patterns encoded by the *OPT-1.3B-One* model during training on two-operator queries. The results of $R^{(l)}$ are averaged over all one-operator queries, and the results of $\kappa^{(l)}$ are averaged over queries from a single one-operator template[8]. We observe that as the accuracy of the *OPT-1.3B-One* model on one-operator queries decreases, $R^{(l)}(\Omega^{\text{opd-opr}})$ and $\kappa_m^{(l)}$ for $m \in \{8, 9, 10\}$ gradually decrease, and most $R^{(l)}$ and $\kappa^{(l)}$ tend to be around 1.0 in the middle layers, **The results validate our Insight 1**. This suggests that the *OPT-1.3B-One* model gradually forgets how to solve simple arithmetic problems while learning more complex knowledge. We think it might be due to spurious forgetting of LLMs (Zheng et al., 2025), which we further discuss in Appendix L.

5 Conclusion

In this paper, we use interactions to provide a deep understanding of the internal mechanism of LLMs for arithmetic reasoning. Through comparison studies of different interaction patterns encoded by LLMs during forward propagation, we find that the internal mechanism of LLMs for solving simple one-operator arithmetic problems is their capability to encode operand-operator interaction patterns and high-order interaction patterns. We further fine-tune an LLM to explore how an LLM encodes different interaction patterns when learning to solve arithmetic problems. We find that the internal mechanism of LLMs for solving relatively complex two-operator arithmetic problems is their capability to encode operator interaction patterns. We also find that an LLM forgets how to solve simple arithmetic problems as it learns to solve relatively complex arithmetic problems.

Limitations

We have only studied simple arithmetic problems and have not yet extended our research to more complex math word problems. In the future, we will work on this.

References

Marco Ancona, Cengiz Oztireli, and Markus Gross. 2019. Explaining deep neural networks with a polynomial time algorithm for shapley value approximation. In *International Conference on Machine Learning*, pages 272–281. PMLR.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*.

Piotr Dabkowski and Yarin Gal. 2017. Real time image saliency for black box classifiers. *Advances in neural information processing systems*, 30.

Huiqi Deng, Qihan Ren, Hao Zhang, and Quanshi Zhang. 2021. Discovering and explaining the representation bottleneck of dnns. *arXiv preprint arXiv:2111.06236*.

Ruth Fong, Mandela Patrick, and Andrea Vedaldi. 2019. Understanding deep networks via extremal perturbations and smooth masks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2950–2958.

Michel Grabisch and Marc Roubens. 1999. An axiomatic approach to the concept of interaction among players in cooperative games. *International Journal of game theory*, 28:547–565.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, et al. 2022. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*.

Scott Lundberg. 2017. A unified approach to interpreting model predictions. *arXiv preprint arXiv:1705.07874*.

Daking Rai and Ziyu Yao. 2024. An investigation of neuron activation as a unified lens to explain chain-of-thought eliciting arithmetic reasoning of llms. *arXiv preprint arXiv:2406.12288*.

Yasaman Razeghi, Robert L Logan IV, Matt Gardner, and Sameer Singh. 2022. Impact of pretraining term frequencies on few-shot reasoning. *arXiv preprint arXiv:2202.07206*.

Jie Ren, Mingjie Li, Qirui Chen, Huiqi Deng, and Quanshi Zhang. 2021a. Towards axiomatic, hierarchical, and symbolic explanation for deep models.

Jie Ren, Mingjie Li, Qirui Chen, Huiqi Deng, and Quanshi Zhang. 2023a. Defining and quantifying the emergence of sparse concepts in dnns. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20280–20289.

Jie Ren, Zhanpeng Zhou, Qirui Chen, and Quanshi Zhang. 2021b. Can we faithfully represent masked states to compute shapley values on a dnn? *arXiv preprint arXiv:2105.10719*.

Qihan Ren, Huiqi Deng, Yunuo Chen, Siyu Lou, and Quanshi Zhang. 2023b. Bayesian neural networks avoid encoding complex and perturbation-sensitive concepts. In *International Conference on Machine Learning*, pages 28889–28913. PMLR.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Lloyd S Shapley. 1953. A value for n-person games. *Contribution to the Theory of Games*, 2.

Alessandro Stolfo, Yonatan Belinkov, and Mrinmaya Sachan. 2023. A mechanistic interpretation of arithmetic reasoning in language models using causal mediation analysis. *arXiv preprint arXiv:2305.15054*.

Mukund Sundararajan, Kedar Dhamdhere, and Ashish Agarwal. 2020. The shapley taylor interaction index. In *International conference on machine learning*, pages 9259–9268. PMLR.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Ben Wang and Aran Komatsuzaki. 2021. Gpt-j-6b: A 6 billion parameter autoregressive language model.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning. *arXiv preprint arXiv:2310.03731*.

Xin Wang, Shuyun Lin, Hao Zhang, Yufei Zhu, and Quanshi Zhang. 2021. Interpreting attributions and interactions of adversarial attacks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1095–1104.

Xin Wang, Jie Ren, Shuyun Lin, Xiangming Zhu, Yisen Wang, and Quanshi Zhang. 2020. A unified approach to interpreting and boosting adversarial transferability. *arXiv preprint arXiv:2010.04055*.

Chris Wendler, Veniamin Veselovsky, Giovanni Monea, and Robert West. 2024. Do llamas work in english? on the latent language of multilingual transformers. *arXiv preprint arXiv:2402.10588*.

Zeping Yu and Sophia Ananiadou. 2024. Interpreting arithmetic mechanism in large language models through comparative neuron analysis. *arXiv preprint arXiv:2409.14144*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Wei Zhang, Chaoqun Wan, Yonggang Zhang, Yiu-ming Cheung, Xinmei Tian, Xu Shen, and Jieping Ye. 2024. Interpreting and improving large language models in arithmetic calculation. *arXiv preprint arXiv:2409.01659*.

Junhao Zheng, Xidi Cai, Shengjie Qiu, and Qianli Ma. 2025. Spurious forgetting in continual learning of language models. *arXiv preprint arXiv:2501.13453*.

Huilin Zhou, Hao Zhang, Huiqi Deng, Dongrui Liu, Wen Shen, Shih-Han Chan, and Quanshi Zhang. 2023. Concept-level explanation for the generalization of a dnn. *arXiv preprint arXiv:2302.13091*.

A Strategies of masking input variables

In the research of attribution methods, it is common to use a specific token or embedding to mask input variables of a DNN (Lundberg, 2017; Ancona et al., 2019; Fong et al., 2019), and use changes in network outputs on the masked samples to estimate attributions of input variables. However, each method has certain limitations. The mean baseline value (Dabkowski and Gal, 2017), i.e., setting the baseline value for each input variable to its mean across all samples, introduces additional signals, e.g., grey dots in images. Similarly, the zero baseline value (Ancona et al., 2019; Sundararajan et al., 2017), i.e., setting baseline values for all input variables to zero, would also introduce additional signals to the input, such as black dots.

B Details about how to mask input words for different LLMs

In this paper, we analyze seven LLMs for arithmetic reasoning, including the OPT-1.3B model, the GPT-J-6B model, the Llama-2-7B model, the CodeLlama-2-7B model, the MathCoder-L-7B model, the MathCoder-CL-7B model, and the Llemma-7B model. For the OPT-1.3B model, we use the "</s>" token with the token $id = 2$ to mask the words in $N\backslash T$. For the GPT-J-6B model, we use the "<|endoftext|>" with the token $id = 50256$ to mask the words in $N\backslash T$. For the other five models, including the Llama-2-7B model, the CodeLlama-2-7B model, the MathCoder-L-7B model, the MathCoder-CL-7B model, and the Llemma-7B model, with Llama as their base model, we use the "<unk>" with the token $id = 0$ to mask the words in $N\backslash T$.

C Properties of interactions

The Harsanyi interaction I_S, i.e., the interaction in this paper, can explain the elementary mechanism of existing game-theoretic metrics (Ren et al., 2021a), including the *Shapley value* (Shapley, 1953), the *Shapley interaction index* (Grabisch and Roubens, 1999), and the *Shapley-Taylor interaction index* (Sundararajan et al., 2020).

(1) *Connection to the Shapley value* (Shapley, 1953). Let $\phi(i)$ denote the Shapley value of an input variable i, given the input sample $\boldsymbol{x}$. Then, the Shapley value $\phi(i)$ can be explained as the result of uniformly assigning attributions of each Harsanyi interaction to each involving variable i, i.e., $\phi(i) = \sum_{S\subseteq N\setminus\{i\}} \frac{1}{|S|+1} I_{S\cup\{i\}}$. This also proves that the Shapley value is a fair assignment of attributions from the perspective of the Harsanyi interaction.

(2) *Connection to the Shapley interaction index* (Grabisch and Roubens, 1999). Given a subset of variables $T \subseteq N$ in an input sample $\boldsymbol{x}$, the Shapley interaction index I_T^{Shapley} can be represented as $I_T^{\text{Shapley}} = \sum_{S\subseteq N\setminus T} \frac{1}{|S|+1} I_{S\cup T}$. In other words, the index I_T^{Shapley} can be explained as uniformly allocating $I_{S'}$ such that $S' = S \cup T$ to the compositional variables of S', if we treat the coalition of variables in T as a single variable.

(3) *Connection to the Shapley Taylor interaction index* (Sundararajan et al., 2020). Given a subset of variables $T \subseteq N$ in an input sample $\boldsymbol{x}$, the k-th order Shapley Taylor interaction index $I_T^{\text{Shapley-Taylor}}$ can be represented as a weighted sum of interaction effects, i.e., $I_T^{\text{Shapley-Taylor}} = I_T$ if $|T| < k$; $I_T^{\text{Shapley-Taylor}} = \sum_{S\subseteq N\setminus T} \binom{|S|+k}{k}^{-1} I_{S\cup T}$ if $|T| = k$; and $I_T^{\text{Shapley-Taylor}} = 0$ if $|T| > k$.

Given an input sample $\boldsymbol{x}$, the Harsanyi interaction I_S satisfies seven desirable axioms in game

theory (Ren et al., 2021a), including the *efficiency, linearity, dummy, symmetry, anonymity, recursive and interaction* distribution axioms.

(1) *Efficiency axiom.* The output score of a model can be decomposed into interaction effects of different patterns, i.e., $v(\boldsymbol{x}) = \sum_{S \subseteq N} I_S$.

(2) *Linearity axiom.* If we merge output scores of two models w and v as the output of model u, i.e., $\forall S \subseteq N, u(\boldsymbol{x}_S) = v(\boldsymbol{x}_S) + w(\boldsymbol{x}_S)$, then their interaction effects $I_S^{(v)}$ and $I_S^{(w)}$ can also be merged as $\forall S \subseteq N, I_S^{(u)} = I_S^{(v)} + I_S^{(w)}$.

(3) *Dummy axiom.* If a variable $i \in N$ is a dummy variable, i.e., $\forall S \subseteq N \backslash \{i\}, v(\boldsymbol{x}_{S \cup \{i\}}) = v(\boldsymbol{x}_S) + v(\boldsymbol{x}_{\{i\}})$, then it has no interaction with other variables, $\forall \emptyset \neq T \subseteq N \backslash \{i\}, I_{T \cup \{i\}=0}$.

(4) *Symmetry axiom.* If input variables $i, j \in N$ cooperate with other variables in the same way, $\forall S \subseteq N \setminus \{i, j\}, v(\boldsymbol{x}_{S \cup \{i\}}) = v(\boldsymbol{x}_{S \cup \{j\}})$, then they have the same interaction effects with other variables, $\forall S \subseteq N \setminus \{i, j\}, I_{S \cup \{i\}} = I_{S \cup \{j\}}$.

(5) *Anonymity axiom.* For any permutations π on N, we have $\forall S \subseteq N, I_S^{(v)} = I_{\pi S}^{(\pi v)}$, where $\pi S \triangleq \{\pi(i) \mid i \in S\}$, and the new model πv is defined by $(\pi v)(\boldsymbol{x}_{\pi S}) = v(\boldsymbol{x}_S)$. This indicates that interaction effects are not changed by permutation.

(6) *Recursive axiom.* The interaction effects can be computed recursively. For $i \in N$ and $S \subseteq N \setminus \{i\}$, the interaction effect of the pattern $S \cup \{i\}$ is equal to the interaction effect of S with the presence of i minus the interaction effect of S with the absence of i, i.e., $\forall S \subseteq N \setminus \{i\}, I_{S \cup \{i\}} = I_S^{(i \text{ is always present})} - I_S$. $I_S^{(i \text{ is always present})}$ denotes the interaction effect when the variable i is always present as a constant context, i.e., $I_S^{(i \text{ is always present})} = \sum_{L \subseteq S} (-1)^{|S|-|L|} \cdot v(\boldsymbol{x}_{L \cup \{i\}})$.

(7) *Interaction distribution axiom.* This axiom characterizes how interactions are distributed for "interaction functions" (Sundararajan et al., 2020). An interaction function v_T parameterized by a subset of variables T is defined as follows: $\forall S \subseteq N, v_T(\boldsymbol{x}_S) = c$, if $T \subseteq S$; otherwise, $v_T(\boldsymbol{x}_S) = 0$. The function v_T models pure interaction among the variables in T because only if all variables in T are present the output value will be increased by c. The interactions encoded in the function v_T satisfy $I_T = c$, and $\forall S \neq T, I_S = 0$.

D Details about network architectures and chosen layers for experiments in section 4

OPT-1.3B model. The OPT-1.3B model is composed of the following parts: one word embedding layer (namely *Embedding Layer*), one position embedding layer (namely *OPTLearnedPositionalEmbedding*), 24 OPT decoder modules (namely *OPTDecoderLayer*), and one linear output layer (namely *Linear Layer*). The architecture can be summarized as *Embedding Layer* $\rightarrow$ *OPTLearnedPositionalEmbedding* $\rightarrow$ [*OPTDecoderLayer*]$\times$24 $\rightarrow$ *Linear Layer*. Each module of *OPTDecoderLayer* contains a self-attention mechanism layer (namely *OPTAttention*), an activation function (namely *ReLU*), a layer normalization operation (namely *LayerNorm*), two fully connected layers (namely *Linear*), and a final layer normalization operation (namely *LayerNorm*). In our experiments, we selected all 24 *OPTDecoderLayer* modules and conducted experiments based on the output features of the *LayerNorm* operation.

GPT-J-6B model. The GPT-J-6B model includes a single word embedding layer (namely *Embedding Layer*), followed sequentially by 28 Transformer blocks (namely *GPTJBlock*), culminating in an output layer normalization (namely *LayerNorm*) and a linear output layer (namely *Linear Layer*). This architecture can be succinctly described as: *Embedding Layer* $\rightarrow$ [*GPTJBlock*]$\times$28 $\rightarrow$ *LayerNorm* $\rightarrow$ *Linear Layer*. Delving into the details of each module of *GPTJBlock*, it comprises three integral components, including a layer normalization (namely *LayerNorm*), a self-attention mechanism (namely *GPTJAttention*), and a feed-forward network (namely *GPTMLP*). In our experiments, we selected all 28 *GPTJBlock* modules and conducted experiments based on the output features of the *LayerNorm* operation.

Other Llama-based models. The other five models based on Llama include the Llama-2-7B model, the CodeLlama-2-7B model, the MathCoder-L-7B model, the MathCoder-CL-7B model, and the Llemma-7B model. These models share the same architecture, which is composed of various elements: a single word embedding layer (namely *Embedding Layer*), followed by a sequence of 32 *LlamaDecoderLayer*, an output layer normalization layer (namely *LlamaRMSNorm*), and culminating in a linear output layer

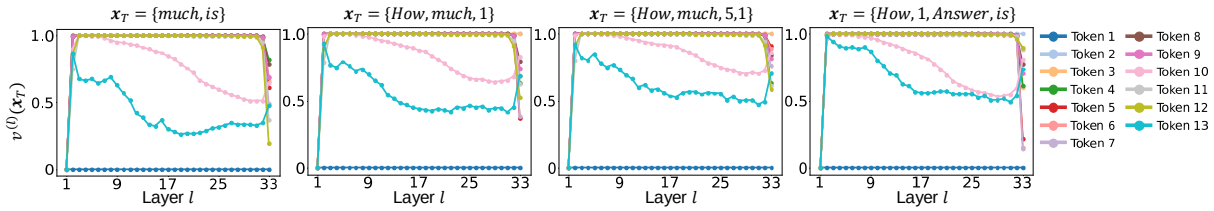

Figure 8: The visualization of $v^{(l)}$ at different token positions across various layers of CodeLlama for the prompt "How much is 5 plus 1? Answer is" under different masked inputs. Results show that except for the final tokens of the complete sentence (i.e., token 10 and token 13), the $v^{(l)}$ values of other tokens in the middle layers remain almost unchanged.

(namely *Linear Layer*). This architecture can be summarized as *Embedding Layer* $\rightarrow$ [*LlamaDecoderLayer*]$\times$32 $\rightarrow$ *LlamaRMSNorm* $\rightarrow$ *Linear Layer*. Delving into each module of *LlamaDecoderLayer*, it integrates multiple components, including a self-attention mechanism layer (namely *LlamaAttention*), a feed-forward network layer (namely *LlamaMLP*) encompassing several linear layers and activation functions, an input layer normalization (namely *Input LayerNorm*), and a post-attention layer normalization (namely *Post Attention LayerNorm*), the latter serving to normalize the output from the self-attention mechanism layer. In our experiments, we selected all 32 modules of *LlamaDecoderLayer* and conducted experiments based on the output features of the *LayerNorm* operation.

E The features of other tokens in the middle layers remain unchanged

Through experiments, we found that, except for the final token of the complete sentence, the $v^{(l)}$ values at other token positions in the middle layers remain almost unchanged. Figure 8 shows the $v^{(l)}$ values at different token positions across various layers of CodeLlama for the prompt "How much is 5 plus 1? Answer is" under different masked inputs x_T. The prompt consists of 13 tokens {<s>, How, much, is, space, 5, plus, space, 1, ?, answer, is, space}. Except for token 10 (?) and token 13 (space), the $v^{(l)}$ values of other tokens in the middle layers remain almost unchanged.

F Prompt Templates

In Table 2 and 3, we report the question templates used as prompts for the model for one- and two-operator queries, respectively. For two-operator queries, we use one query template for each of the 29 possible two-operation combinations. To enable the model to output the answer directly, we appended "The answer is" at the end of each template.

G Performance of the LLMs

In Tables 4 and 5, we report the accuracy of the LLMs on the arithmetic queries used in our analyses. For the OPT-1.3B and GPT-J-6B models, we treat each operand as a single token, while for other LLMs, each number is split into multiple tokens ("0", "1", "2", . . . , "9") by the tokenizer.

H Fine-tuning Details

We fine-tune the OPT-1.3B model using the LoRA architecture on one-operator and two-operator templates. For the one-operator templates, the model is trained for 10 epochs on a dataset consisting of 3,500 samples with a batch size of 16. For the two-operator templates, we train the model for 20 epochs on a dataset containing 29,000 samples, with a batch size of 32. The training uses a learning rate of 8e-4 with a linear decay scheduler. The LoRA configuration includes a rank of 8, a LoRA alpha of 32, and a dropout of 0.05. We ensure no overlap between the training data and any evaluation or testing datasets.

I More experimental results

Figure 9 shows that LLMs with good arithmetic capabilities gradually focus more on high-order interaction patterns, while LLMs with poor arithmetic capabilities exhibit a more dispersed focus across interaction patterns of different orders in the middle layers. Figure 10 illustrates that the OPT-1.3B model gradually increases its focus on high-order interaction patterns during the learning process of simple arithmetic problems. Figure 11 demonstrates that the OPT-1.3B model gradually decreases its focus on high-order interactions while learning relatively complex arithmetic problems.

Type	Addition	Subtraction
1	How much is n_1 plus n_2? Answer is	How much is n_1 minus n_2? Answer is
2	What is n_1 plus n_2? Answer is	What is n_1 minus n_2? Answer is
3	How much is the sum of n_1 and n_2? Answer is	How much is the difference between n_1 and n_2? Answer is
4	What is the sum of n_1 and n_2? Answer is	What is the difference between n_1 and n_2? Answer is
5	The sum of n_1 and n_2 is	The difference between n_1 and n_2 is
6	Given two numbers n_1 and n_2, the sum of them is	Given two numbers n_1 and n_2, the difference between them is
	Multiplication	**Division**
1	How much is n_1 times n_2? Answer is	How much is n_1 over n_2? Answer is
2	What is n_1 times n_2? Answer is	What is n_1 over n_2? Answer is
3	What is the result of n_1 times n_2? Answer is	What is the result of n_1 over n_2? Answer is
4	How much is the product of n_1 and n_2? Answer is	How much is the ratio between n_1 and n_2? Answer is
5	The product of n_1 and n_2 is	The ratio of n_1 and n_2 is
6	Given two numbers n_1 and n_2, the product of them is	Given two numbers n_1 and n_2, the ratio between them is

Table 2: Question templates for one-operator arithmetic queries.

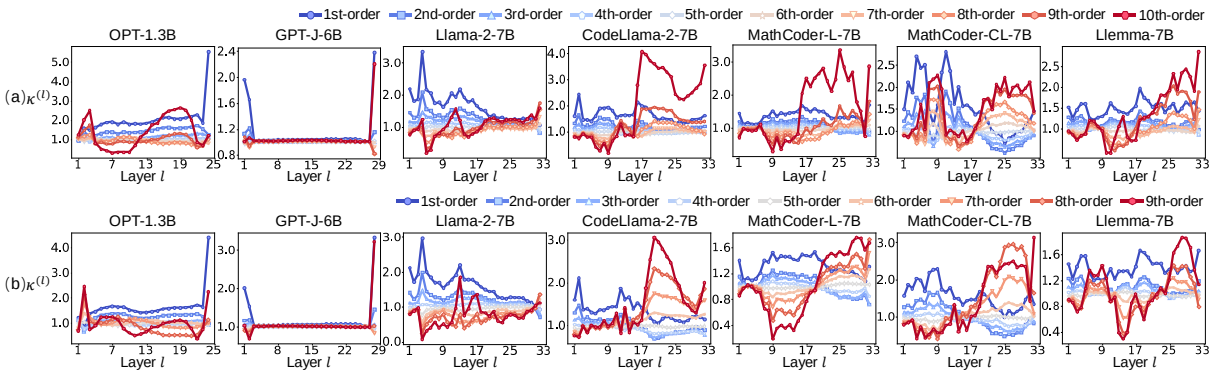

Figure 9: Comparing the normalized average strength $\kappa^{(l)}$ of interaction patterns of different orders encoded by LLMs during forward propagation. Each curve in the figure is averaged over various one-operator arithmetic queries, corresponding to (a) template 2 and (b) template 3 in Table 2.

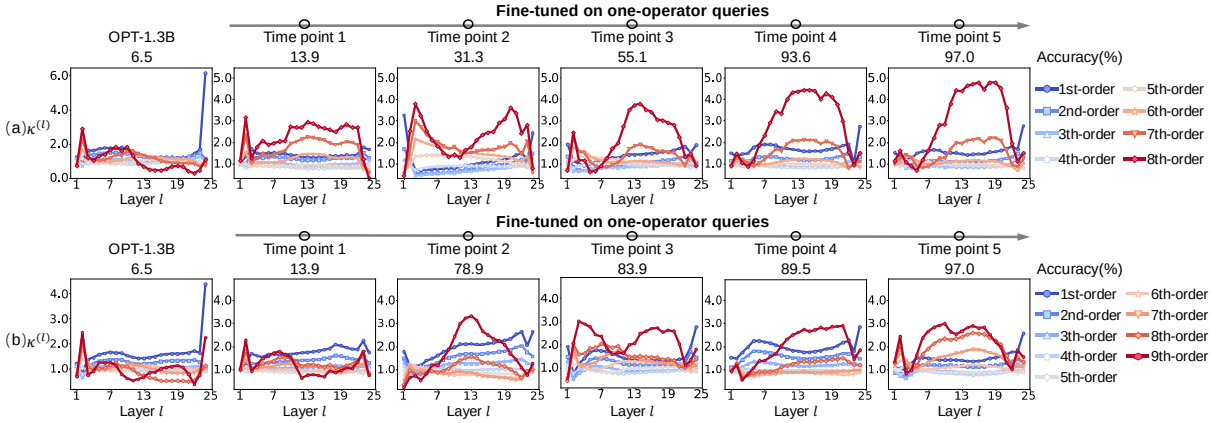

Figure 10: Visualizing the dynamic process of different interaction patterns encoded by the OPT-1.3B model during the training process. Each curve is averaged over various one-operator queries, corresponding to (a) template 0 and (b) template 3 in Table 2.

Formula	Format
$f = ((A + B) * C)$	Sum A and B and multiply by C
$f = (A + B * C)$	What is the sum of A and the product of B and C?
$f = ((A - B) * C)$	What is the product of A minus B and C?
$f = (A/(B/C))$	How much is A divided by the ratio between B and C?
$f = (A * (B - C))$	What is the difference between A and the product of B and C?
$f = ((A + B))$	How much is A times the difference between B and C?
$f = (A - (B - C))$	What is the ratio between A plus B and C?
$f = ((A - B))$	How much is A minus the difference between B and C?
$f = ((A - B * C))$	What is the ratio between A minus B and C?
$f = (A - B/C)$	What is the difference between A and the ratio between B and C?
$f = (A/(B + C))$	How much is A divided by the sum of B and C?
$f = (A/(B - (C)))$	How much is A divided by the difference between B and C?
$f = ((A + B)/C)$	What is the sum of A and the ratio between B and C?
$f = (A * (B/C))$	How much is A times the ratio between B and C?
$f = (A * (B + C))$	How much is A times the sum of B and C?
$f = (A + B/C)$	How much is the sum of A divided by B and C?
$f = (A/(B/C))$	How much is A divided by B divided by C?
$f = (A * (B/C))$	How much is the difference between A divided by B and C?
$f = (A/B)$	How much is A divided by B times C?
$f = (A - (B * C))$	How much is A divided by B times C?
$f = (A + B * C)$	How much is A divided by C?
$f = (A + B * C)$	How much is A plus B times C?
$f = (A/(B + C))$	How much is A times B times C?

Table 3: Question templates for two-operator arithmetic queries.

1-opr	OPT-1.3B	GPT-J-6B	Llama-2-7B	CodeLlama-2-7B	MathCoder-L-7B	MathCoder-CL-7B	Llemma-7B
2	0.071	0.295	0.941	0.887	0.920	0.843	0.979
3	0.020	0.368	0.637	0.773	0.803	0.707	0.886
4	0.123	0.205	0.670	0.801	0.762	0.663	0.917
5	0.057	0.125	0.701	0.741	0.769	0.733	0.844
6	0.079	0.071	0.847	0.648	0.845	0.744	0.895
7	0.106	0.431	0.733	0.707	0.545	0.475	0.707

Table 4: Accuracy(%) of 7 models on 6 one-operator templates.

J Information about the use of AI assistants.

In this paper, AI tools such as DeepSeek were used for translation and grammar checking.

K Computational budget

We conducted our experiments on an NVIDIA GeForce RTX 3090 24GB GPU. For the Llama-2-7B model, the computation time per one-operator sample is around 30 seconds, while that for a two-operator sample is around 60 seconds.

2-opr	OPT-1.3B	GPT-J-6B	Llama-2-7B	CodeLlama-2-7B	MathCoder-L-7B	MathCoder-CL-7B	Llemma-7B
1	0.00	0.010	0.004	0.007	0.004	0.003	0.007
2	0.000	0.001	0.015	0.016	0.001	0.005	0.015
3	0.012	0.029	0.016	0.023	0.013	0.027	0.022
4	0.078	0.056	0.117	0.124	0.132	0.193	0.128
5	0.045	0.031	0.051	0.061	0.047	0.045	0.098
6	0.002	0.027	0.044	0.051	0.044	0.036	0.054
7	0.117	0.152	0.144	0.160	0.157	0.157	0.168
8	0.004	0.009	0.028	0.032	0.026	0.025	0.028
9	0.116	0.203	0.328	0.446	0.257	0.282	0.424
10	0.018	0.008	0.036	0.041	0.019	0.028	0.048
11	0.195	0.360	0.161	0.157	0.159	0.163	0.363
12	0.221	0.175	0.372	0.368	0.369	0.367	0.373
13	0.005	0.002	0.074	0.073	0.021	0.016	0.034
14	0.046	0.116	0.143	0.144	0.104	0.146	0.229
15	0.000	0.025	0.027	0.025	0.007	0.015	0.053
16	0.000	0.009	0.063	0.055	0.004	0.012	0.048
17	0.018	0.004	0.004	0.016	0.021	0.024	0.051
18	0.200	0.460	0.607	0.471	0.492	0.623	0.554
19	0.022	0.066	0.142	0.168	0.256	0.192	0.205
20	0.035	0.062	0.102	0.146	0.122	0.149	0.223
21	0.021	0.035	0.36	0.107	0.075	0.092	0.272
22	0.000	0.013	0.019	0.020	0.010	0.020	0.028
23	0.235	0.308	0.517	0.270	0.339	0.304	0.644
24	0.004	0.011	0.030	0.064	0.025	0.048	0.081
25	0.000	0.015	0.581	0.163	0.355	0.128	0.559
26	0.004	0.100	0.431	0.176	0.152	0.118	0.361
27	0.059	0.142	0.132	0.233	0.126	0.213	0.361
28	0.002	0.030	0.058	0.065	0.042	0.057	0.104
29	0.005	0.059	0.237	0.152	0.068	0.113	0.000

Table 5: Accuracy(%) of 7 models on 29 two-operator templates.

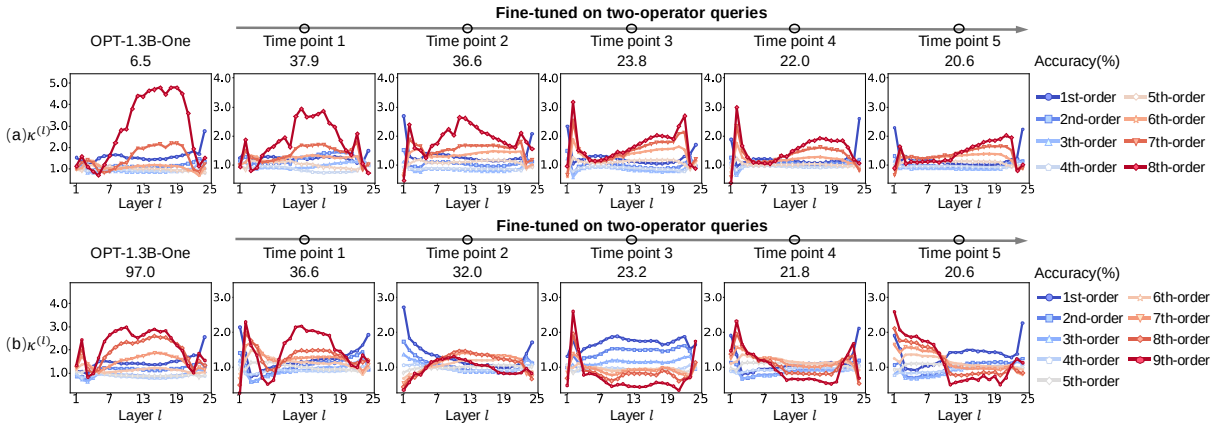

Figure 11: Visualizing the dynamic process of different interaction patterns encoded by the *OPT-1.3B-One* model during the training process. Each curve is averaged over various one-operator queries, corresponding to (a) template 0 and (b) template 3 in Table 2.

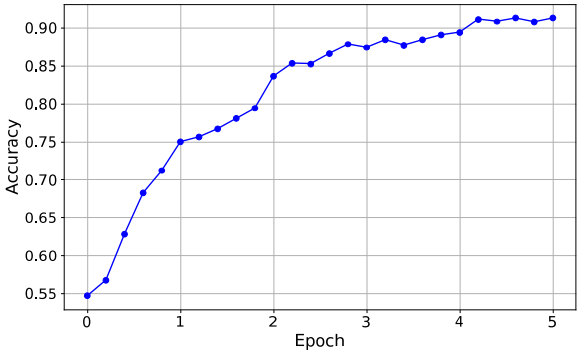

Figure 12: The change in accuracy of *OPT-1.3B-Both* on one-operator queries during the training process.

L Spurious Forgetting of LLMs

We further train the *OPT-1.3B-One* model on two-operator queries. This version is termed the *OPT-1.3B-Both* model. The results in Section 4.2 show that the accuracy of the *OPT-1.3B-One* model on one-operator problems declines during the training process on two-operator queries. Following (Zheng et al., 2025), we further fine-tune the *OPT-1.3B-Both* model on half of the training data from one-operator queries used to train the *OPT-1.3B-One* model (see Appendix H). Figure 12 shows the accuracy of *OPT-1.3B-Both* on one-operator queries during the training process. We observe that after just 3 epochs, the accuracy reaches 0.85, and at the fifth epoch, it reaches 0.90. Therefore, we think the performance loss of the *OPT-1.3B-One* model on one-operator queries during the training process on two-operator problems might be due to spurious forgetting. That is, the performance loss does not necessarily indicate a loss of knowledge, but rather a decline in task alignment.