Implicit Reasoning in Transformers is Reasoning through Shortcuts

Tianhe Lin[♡], **Jian Xie**[♠], **Siyu Yuan**[♡], **Deqing Yang**[♡*]
[♡]School of Data Science, Fudan University
[♠]College of Computer Science and Artificial Intelligence, Fudan University
{thlin20,yangdeqing}@fudan.edu.cn
{jianxie22, syyuan21}@m.fudan.edu.cn

Abstract

Test-time compute is emerging as a new paradigm for enhancing language models' complex multi-step reasoning capabilities, as demonstrated by the success of OpenAI's o1 and o3, as well as DeepSeek's R1. Compared to explicit reasoning in test-time compute, implicit reasoning is more inference-efficient, requiring fewer generated tokens. However, ***why does the advanced reasoning capability fail to emerge in the implicit reasoning style?*** In this work, we train GPT-2 from scratch on a curated multi-step mathematical reasoning dataset and conduct analytical experiments to investigate how language models perform implicit reasoning in multi-step tasks. Our findings reveal: 1) Language models can perform step-by-step reasoning and achieve high accuracy in both in-domain and out-of-domain tests via implicit reasoning. However, this capability only emerges when trained on fixed-pattern data. 2) Conversely, implicit reasoning abilities emerging from training on unfixed-pattern data tend to overfit a specific pattern and fail to generalize further. Notably, this limitation is also observed in state-of-the-art large language models. These findings suggest that language models acquire implicit reasoning through shortcut learning, enabling strong performance on tasks with similar patterns while lacking generalization. Resources are available on the GitHub.

1 Introduction

Chain-of-Thought (CoT; Wei et al. (2022)) has sparked the development of explicit reasoning in large language models (LLMs). The subsequent rise of large reasoning models (OpenAI, 2024b; Google, 2024; DeepSeek-AI, 2025) based on long CoT demonstrates impressive capabilities across various tasks (Rein et al., 2023; MAA, 2024; Jimenez et al., 2024). Recent works have shown that such reasoning capabilities can even be distilled into smaller models (DeepSeek-AI, 2025).[1] Different from explicit reasoning, implicit reasoning offers greater inference efficiency by relying on fewer tokens to generate an answer (Deng et al., 2023). Yet, it falls short of the performance achieved by explicit reasoning (Deng et al., 2024; Allen-Zhu and Li, 2024). *Why can't implicit reasoning develop advanced reasoning capabilities?*

While recent advances in mechanistic interpretability have aimed to demystify the implicit reasoning processes of language models (LMs), most studies are limited to single-step reasoning (Meng et al., 2022; Wang et al., 2023; Nanda et al., 2023), which does not meet the expectation for handling complex reasoning tasks, such as advanced mathematical problems. Meanwhile, for multi-step implicit reasoning, previous work primarily focuses on reasoning over factual knowledge (Yang et al., 2024a; Biran et al., 2024), which may be hindered by issues such as inflated reasoning performance due to memorizing entity co-occurrences in the pretraining data (Elazar et al., 2023; Kang and Choi, 2023; Ju et al., 2024).

In this paper, to minimize the impact of memorization and investigate the underlying reasoning mechanisms, we explore implicit reasoning through the lens of mathematical problems. Mathematical reasoning primarily depends on arithmetic operations that follow strict logical rules, which require algebraic manipulation based on specific operators and operands rather than recalling pre-trained knowledge like entity relationships. Given that the strength of explicit reasoning stems from stepwise rationales, the first question we seek to address is **RQ1: Can language models perform stepwise reasoning internally?** To investigate this, we train GPT-2 from scratch on our synthetic multi-

*Corresponding author.

[1]In this paper, "smaller" is relative to super large LMs like Deepseek R1, which has 671B parameters.

Findings of the Association for Computational Linguistics: ACL 2025, pages 9470–9487

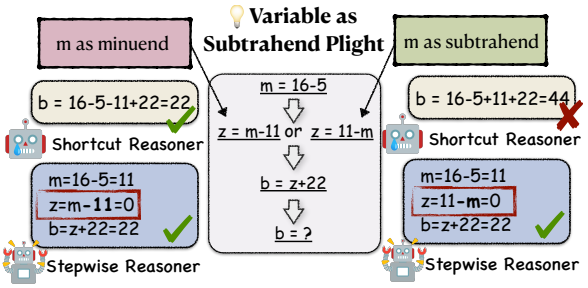


Figure 1: A failure of generalization in language models trained on data with unfixed patterns, namely "Variable as Subtrahend Plight". When trained on unfixed premise order, the model learns a reasoning shortcut that benefits from addition commutativity. This shortcut enables the model to perform implicit reasoning by chaining numbers, which fails when variables are subtrahends.

step dataset composed of sequential mathematical operations, which means premises are arranged in the same order as they appear in the actual step-by-step calculation process. The experimental results and activation patching (Vig et al., 2020; Meng et al., 2022) plots show that **LMs can fully learn to do stepwise reasoning internally and generalize to problems with more steps, provided they are trained on data where all premises are presented sequentially.**

However, the premises are not always presented sequentially in real-world reasoning tasks, requiring LMs to organize the information internally. Therefore, based on the findings from RQ1, this paper seeks to answer a more general research question, **RQ2: How do language models think internally if the premise order is not fixed?** In contrast to the accuracy saturation in RQ1, accuracy drops significantly when the premise order is unfixed. We conduct further analysis and find **LMs fail to learn stepwise implicit reasoning when the premise order is not fixed, struggling with "Variable as Subtrahend Plight".** Specifically, as shown in Figure 1, models trained on an unfixed premise order overfit to an easy pattern in the data, relying on a shortcut that benefits from addition commutativity. This shortcut allows the model to solve the problem by directly chaining numbers, while the presence of variables in the subtrahend position disrupts this shortcut. Additional mechanistic analysis validates our hypothesis.

Previous work demonstrated that even current state-of-the-art (SoTA) LLMs also struggle with implicit reasoning (Yu, 2024). Based on our previous findings, we aim to investigate **RQ3: How do LLMs perform multi-step implicit reasoning?** We find **"Variable as Subtrahend Plight" also persists in SoTA LLMs, indicating that these models, trained on diverse unfixed premise corpora, are also relying on shortcuts for multi-step implicit reasoning.** This further validates the correctness and generalizability of our findings.

To summarize, in this paper, we investigate the internal mechanisms of implicit reasoning in transformers and uncover why the advanced reasoning capabilities observed in explicit reasoning do not emerge in implicit reasoning. While we reveal that current LMs primarily rely on shortcuts for implicit reasoning, a silver lining is that a stepwise reasoning pattern could indeed emerge through training. Such a pattern underpins the advanced reasoning capabilities of LMs, and we envision that future advanced strategies could help form this pattern.

2 Related Work

2.1 Mechanistic Interpretability of Language Models

Mechanistic interpretability (MI) aims to uncover and explain the internal workings of models. The research of mechanistic interpretability in language models primarily focuses on three key areas: *features* within model representations (nostalgebraist, 2020; Gurnee et al., 2023; Zhou et al., 2024), *circuits* connecting these features (Wang et al., 2023; Hanna et al., 2023; Prakash et al., 2024), and *universality* across diverse models and tasks (Chughtai et al., 2023; Gurnee et al., 2024).

Mathematical tasks, due to their significance in representing the reasoning capabilities of language models, have been widely studied in MI (Hanna et al., 2023; Kudo et al., 2024; Zhou et al., 2024). However, most of the existing studies (Stolfo et al., 2023; Yu and Ananiadou, 2024; Zhang et al., 2024; Chen et al., 2024) focus on single-step mathematical reasoning. How LMs perform multi-step mathematical reasoning implicitly remains poorly understood. To bridge this gap, we employ activation patching (Vig et al., 2020) to track the information flow and reverse-engineer the behaviors of LMs in multi-step arithmetic computations.

2.2 Multi-step Implicit Reasoning

As opposed to explicit reasoning, implicit reasoning is performed in the hidden states instead of extra tokens. Previous studies typically investigate implicit reasoning in two domains: factual reason-

ing (Wang et al., 2024; Yang et al., 2024a,b; Biran et al., 2024) and mathematical reasoning (Stolfo et al., 2023; Nanda et al., 2023; Deng et al., 2024). However, progress in reasoning over factual knowledge risks being inflated by entity co-occurrence learned from pre-training data (Elazar et al., 2023; Kang and Choi, 2023; Ju et al., 2024). While mathematical reasoning is less susceptible to this issue due to the variability of operands and operators, LMs may rely on shortcuts or shallow heuristics to predict the results (Liu et al., 2023; Nikankin et al., 2025; Xie et al., 2024), which are often overlooked in studies on multi-step implicit reasoning. In our study, we scrutinize the impact of shortcuts and represent the internal mechanisms driving the observed phenomenon to the investigation of the multi-step implicit mathematical reasoning abilities in Transformer-based LMs.

3 General Setup

Task. Focusing on reasoning capability rather than other factors (e.g., factual knowledge memorization), we use mathematical problems as a lens. To further minimize the impact of natural language complexity, we shift our focus to mathematical formulas rather than problem statements in natural language. Specifically, we construct a synthetic dataset of multi-step sequential modular addition and subtraction as our testbed for analysis. As shown in Figure 1, except for the first step, each step of the computation involves a variable from the previous step, a number (we name it operand later), and an operator (i.e., "+" or "−"). Following Ye et al. (2024), we consider using arithmetics mod23 to avoid numbers being split into multiple tokens and prevent errors from large number calculations, thereby focusing on reasoning itself rather than calculation.

Data. For training data, we generate different multi-step calculation templates for questions at each length (ranging from 1 to 5 steps) and then randomly use K different groups of variable names to instantiate each template.[2] To prevent LMs from memorizing intermediate results from the training set rather than performing actual reasoning to solve the math problems in our test set, we filter out all templates with preceding calculations, apart from the first step, that overlap with the templates of the training set during the test set generation process. For example, if "f=1+2,s=3-f,s»?" appears in the training set, then "a=1+2,b=3-a,c=b+5,c»?" is not allowed to appear in the test set because the first two steps of the former are the same as the latter regardless of variable names.

We evaluate both in-distribution (ID) and out-of-distribution (OOD) performance, which are distinguished by the maximum reasoning steps of the training set, with ID not exceeding the maximum steps of the training set (i.e., 5-step) and OOD being one or two steps more than the maximum steps of the training set (i.e., 6-step or 7-step). ID generalization aims to evaluate whether the model learns the latent rules of the training set, while OOD generalization is designed to assess whether the model genuinely acquires some reasoning skills.

Model & Optimization. Following Ye et al. (2024), we use a standard 12-layer GPT-2 model (Radford et al., 2019) and replace its positional embeddings with rotary embeddings (RoPE) (Su et al., 2024) to enable the model to learn length generalization (i.e., to generalize its ability to solve more steps in multi-step reasoning tasks than those seen during training). We use AdamW (Loshchilov and Hutter, 2019) with learning rate 10^{-4}, batch size 1600, weight decay 0.1 and 2000 warm-up steps.

Activation Patching. Activation patching (Vig et al., 2020; Meng et al., 2022) is a strategy for identifying the important modules that causally affect the output by intervening on their latent activations. Specifically, if a module is important, the alteration of its activation will significantly affect the model's output, whereas an unimportant one will have little to no impact. Typically, the method needs two inputs, an original one (e.g., "a=1+4,d=5-a,c=1+d,c»?") and another with a slight difference (e.g., "a=6+4,d=5-a,c=1+d,c»?"), and three forward passes: The **clean run** and the **corrupted run** take the above two inputs separately and cache activations of the model's components, such as attention or MLP outputs. In the **patched run**, we run the model on the original input but replace the specific activation with the cached activation from the corrupted run. Following previous work (Zhang et al., 2024), we measure the changes in the output logits of the ground truth tokens. Then, we compute the patching effect (PE) as:

$$\mathrm{PE} = \frac{\mathrm{Logit}_{\mathrm{cl}}(r) - \mathrm{Logit}_{\mathrm{pt}}(r)}{\mathrm{Logit}_{\mathrm{cl}}(r)}, \quad (1)$$

[2] $K = 2$ in this paper. Please refer to Appendix A for more details about the data generation process.

where r is the correct answer of the original input and cl, pt denote the clean and patched run separately. The experiments are conducted on 100 randomly selected samples.

We iterate activation patching over a set of activations and compare how much they affect the final output, which allows us to localize which activation matters and ultimately reverse-engineer the underlying circuit. In practice, we utilize sliding window patching (Hase et al., 2023) with window size 2×2, where at each token position, the representations of the 2×2 region formed by the current layer and the next layer, along with the current token and the next token, are substituted by the cached activations from the corrupted run.[3]

4 Can Language Models Perform Stepwise Reasoning Internally?

Previous work found that smaller LMs ($\sim$7B) can hardly do multi-step mathematical reasoning correctly without CoT, while a 70B level model can only achieve an accuracy of about 50% in 4-hop reasoning (Yu, 2024). Since previous work demonstrated that externalizing reasoning step by step enhances performance in mathematical tasks (Wei et al., 2022), a question is: *does the poor performance of implicit reasoning arise from the inability to employ this step-by-step reasoning style?* We begin our investigation by training our GPT-2 model on the synthetic dataset to learn implicit reasoning.

4.1 Results

Language models are able to perform implicit mathematical reasoning with near-complete accuracy when trained. We first analyze whether our model is capable of solving multi-step implicit mathematical reasoning. Figure 2 shows the model's accuracy on both the ID and OOD test data throughout the optimization. The model not only achieves 100% accuracy on implicit reasoning tasks from the same distribution (ID set) but also ***generalizes effectively to tasks requiring longer reasoning steps in the OOD set***. To be specific, the model achieves 99% accuracy in tasks that require an additional step of reasoning and nearly 90% accuracy in tasks that require two more. This implies that the model truly learns some implicit reasoning skills rather than simply memorizing answers, since our model has never seen any training example of the same length as in the test time.

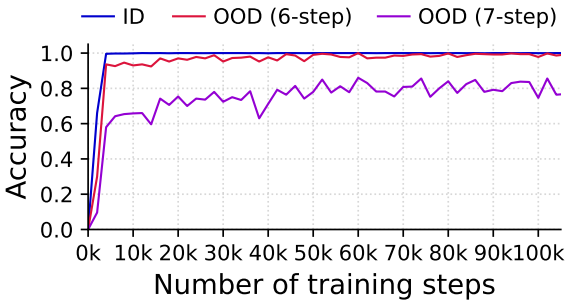


Figure 2: Test accuracy during the training stage. We find that Transformers are able to learn to reason implicitly and generalize well to those that require longer reasoning steps.

4.2 The Working Mechanism of Model

Setting. To investigate whether the language model is based on understanding (i.e., gathering all the information together first and then computing) or reasoning step by step, we use activation patching to reveal the model's internal thought process. Two different experiment setups are used to reveal how the information is transmitted and what information is transmitted separately.

• ***Tracing the information flow.*** To gain insights into the working mechanisms of the model, we first need to know the path through which the token's information is transmitted to the output, i.e., how the information of a specific token affects the output. To this end, we change only one operand or operator in the original input and identify the activations that have an influence on the final output by replacing activations.

• ***Tracking result-related information.*** The first setting explains how information is transmitted to the output, yet what information is transmitted is still unclear. Therefore, we formulate a variant of the first setting to track the information related to intermediate results (i.e., the value of an intermediate variable). Specifically, we modify a set of operands and compare the differences in patching effects when the intermediate results are either identical or distinct. For example, if we aim to track the related information of "d" in "a=4+6,**d**=a+5,c=1+d,c»?", We need to modify the operands while keeping the value of "d" fixed at 15 (e.g., "a=4+1, **d**=a+10,c=1+d,c»?") and compare it with a case where the result changes (e.g., "a=4+1,**d**=a+4,c=1+d,c»?", where d=9). If the model is performing step-by-step reasoning, the patching effect will be more pronounced in the first two steps due to the change in "a" and

[3]We study the choice of metrics in Appendix B.1 and window sizes in Appendix B.2.

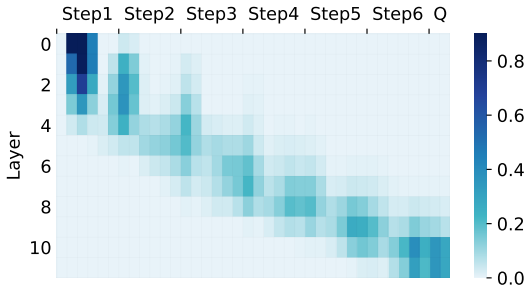


Figure 3: Activation patching on residual stream across layers and token positions when changing the first number in the problems. All the premise orders are forward.

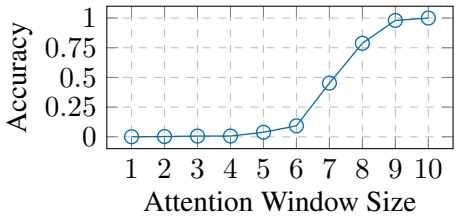


Figure 4: Test accuracy under different attention window sizes on 5-step problems. A window size of n means that a token can focus on itself and its preceding $n-1$ tokens.

will then diminish from the third step onward in the fixed-result setting, as the subsequent results remain unchanged.

Language models are able to do step-by-step reasoning internally. To trace the information flow, we first examine the residual stream patching plot by only altering one operand.[4] The patching effects across layers and positions are shown in Figure 3. We observe that a significant portion of the patching effects concentrate at the end of each step and exhibit a clear trend of gradually propagating along a diagonal line. This pattern forms the foundation of step-by-step reasoning, which implies that each intermediate result builds upon the last.

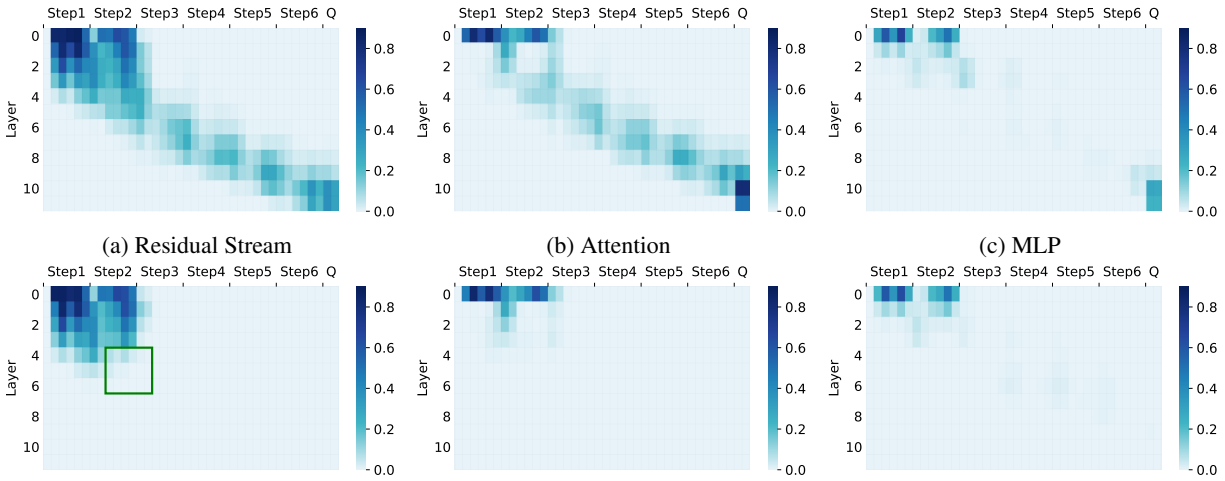
Based on the discovered information flow, we investigate the information behind these activations by tracking result-related information and adding constraints to ensure the results remain the same when changing the input. By comparing the results of the result-varied setting (Figure 5a) and the result-fixed setting (Figure 5d), we find: The region between Step 2 and Step 3, where the impact diminishes (highlighted by the green box in Figure 5d), aligns precisely with the segment between the second and third steps in the information flow (Figure 3). In the fixed-result setting, the substituted activations retain the same information, leading to a minor patching effect. However, in the unfixed setting, the patching effect is more pronounced. This provides evidence that this area stores information related to the intermediate results.

To further validate that the language model performs step-by-step reasoning by reusing intermediate results stored in a specific area from the last step, we conduct an additional experiment to examine its behavior when the information from previous steps is masked by varying the attention window size (see implementation details in Appendix D). Specifically, each step consists of 6 tokens, and when we scale up the attention window size to bigger than 6, the model is able to access the information stored from the previous step. We present the model's accuracy under different attention window sizes in Figure 4. Our findings show that when the attention is restricted to the current step (i.e., window size = 6), the model completely loses its reasoning ability. However, once the attention is expanded to include the previous step's results, accuracy recovers rapidly. This supports the hypothesis that the model follows a step-by-step reasoning pattern, as evidenced from a different perspective.

To sum up, the model computes the result of each step once it concludes, and this information is then utilized by the subsequent step in the next layers, establishing a step-by-step computation pattern.

Attention mechanism propagates intermediate results, and MLP modules enhance features related to inputs and outputs. Intervening on hidden states only provides us with a glimpse of the information flow, but the roles of various components within the model remain unclear. By decomposing the causal effects of contributions of attention and MLP modules (Figure 5b,5e and Figure 5c,5f), we find a decisive role for attention modules in the middle layers and MLPs in early and final layers. In conjunction with the findings on information flow, we infer that the attention layers are responsible for extracting the information needed in the current step and gradually transferring intermediate computational information to deeper layers. Therefore, a possible explanation of the model's behavior on this task is that the MLP modules enhance features of operators and operands in early layers, then the attention mechanism facilitates the step-by-step propagation of intermediate results, and finally, the MLP modules in the last layers enhance the probabilities of correct predictions.

[4]We show the information flow related to operators in Appendix C.



(a) Residual Stream (b) Attention (c) MLP

(d) Residual Stream (e) Attention (f) MLP

Figure 5: Patching effect of different components across layers and token positions. We change the numbers in the first two steps. The result of step 2 is **changed** in sub-figure (a)(b)(c), while the result is kept **unchanged** in sub-figure (d)(e)(f). A deeper color indicates the significance of activation at that position. We add a green rectangle in the figure to better illustrate the location where the patching effect first starts to diminish.

5 How Do Language Models Reason Internally When the Premise Order is Not Fixed?

Based on the above findings, we find that Transformers are able to perform step-by-step reasoning internally when the premise order is fixed. However, in complex reasoning tasks, the premises are not always presented sequentially; they may appear in a random order, requiring LMs to organize the information internally. *Can language models still perform implicit reasoning step by step when the premises are shuffled?*

Setup. For consistency, we continue to use the original data but randomly shuffle the order of premises[5], excluding the question. To assess the impact of premise order, we study three different orders, including forward, reverse, and random. Specifically, for the reverse order, we list the premises in reverse order; for the random order, we shuffle the premises randomly.

5.1 Result

Language models fail to learn implicit reasoning when the premise order is not fixed. In contrast to the high accuracy scores achieved by the model trained on fixed-order premises, Table 1 shows that the model trained on shuffled premises fails to perform multi-step implicit reasoning correctly. Specifically, as the number of steps increases, the model's accuracy gradually decreases, reaching only ∼40% accuracy when five steps of reasoning are required, contrasting the saturated accuracy of the model trained on fixed premise order.

Order	2-Step	3-Step	4-Step	5-Step	6-Step
Forward	1.00	0.87	0.57	0.43	0.23
Reverse	1.00	0.81	0.51	0.38	0.19
Random	1.00	0.83	0.53	0.37	0.23

Table 1: The accuracy of the model trained with unfixed premise order dataset on the original test set. Each column represents problems with a specific number of steps, and each row represents a premise order used during testing.

Language models struggle with "Variable as Subtrahend Plight." To explore how LMs perform implicit reasoning after being trained on an unfixed premise order, we conduct further analysis and find that the model is more prone to making mistakes when the premise contains multiple equations with a variable as the subtrahend. Detailed statistics are provided in Table 2 on how the model's accuracy varies with the number of variables being subtrahends for questions requiring three to five steps of reasoning. As the number of variables being subtracted increases, the model's accuracy decreases drastically, which is consistent across different premise orders. We term this phenomenon as "Variable as Subtrahend Plight." When almost all the variables in the premise are subtrahends, the model almost fails to solve any of the problems correctly. To rule out the possibility of a special case, we conduct experiments with in-

[5]More details of data setups are included in Appendix E.

Order	#Variable Being Subtrahend				
	0	1	2	3	4
3-Step Problems					
Forward	1.00	0.83	0.35	-	-
Reverse	1.00	0.73	0.15	-	-
Random	1.00	0.73	0.26	-	-
4-Step Problems					
Forward	1.00	0.33	0.08	0.12	-
Reverse	1.00	0.15	0.08	0.10	-
Random	1.00	0.23	0.08	0.08	-
5-Step Problems					
Forward	0.92	0.20	0.04	0.05	0.03
Reverse	0.90	0.10	0.04	0.03	0.03
Random	0.90	0.09	0.03	0.02	0.02

Table 2: Accuracy of the model on questions with different numbers of variables being subtrahends. The accuracy is calculated on 100 instances. Since the first step involves an operation between two numbers, the maximum number of variables as subtrahends is one less than the total number of steps.

creased data volume and with different models, yet the phenomenon remains consistent. Please refer to Appendix F for more details.

To explore why models struggle with the "Variable as Subtrahend Plight", we revisit arithmetic expressions. While addition benefits from commutativity (e.g., a+b=b+a), subtraction lacks this property, as swapping the minuend and subtrahend changes the outcome unless a=b. This asymmetry creates challenges for models. For instance, in the sequence "a=6+2,b=a-3,c=4+b", the model might shortcut it as "c=6+2-3+4" (treating subtraction as addition). However, when subtrahends are variables, such a shortcut fails. If "b=3-a", the model can no longer chain terms directly and must compute intermediate results in sequence. As the number of variable subtrahends increases, the model faces greater difficulty in determining the correct order of operations, requiring rigorous step-by-step reasoning instead of relying on shortcuts.

Language models do not think step-by-step when the premise order is not fixed and overfit to an incorrect shortcut. Based on our analysis above, accuracy sharply declines if the model relies on shortcut computation. In contrast, step-by-step computation would result in minimal accuracy variation, as whether variables are subtrahends or not does not significantly affect sequential reasoning. To validate our hypothesis, we plot the accuracy trends against the number of equations

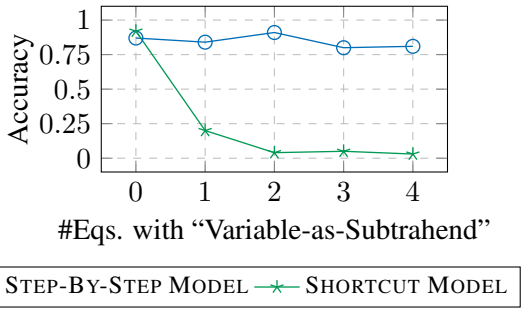


Figure 6: Test accuracies with increasing number of equations containing a variable as the subtrahend. The step-by-step computation model (Step-by-step Model) is evaluated on OOD 7-step problems since the accuracy of this model in both ID ones and OOD 6-step is nearly 100%. The model trained on problems with unfixed premise order (Shortcut Model) is evaluated on ID 5-step problems.

with "Variable-as-Subtrahend" in our step-by-step computation model used in Section 4. As shown in Figure 6, there is only a slight variation in the accuracy of the step-by-step computation model while the accuracy of the model trained on problems with varied premise order drops significantly, which verifies our hypothesis.[6]

To sum up, when the training data follows a fixed pattern, LMs can learn a fixed pattern to store each intermediate result upon completing a step. For instance, in a forward premise order, the model simply follows the operators to compute the results of operands sequentially (i.e., step-by-step reasoning). There is no need to track the variables, as they must come from the previous step. However, when the premise order is shuffled, this shortcut pattern no longer exists, which necessitates the true reasoning capability: first tracking the variables and then performing the computation. More steps involve more complex tracking and computation, which explains why accuracy decreases as the number of steps increases. This implies that when LMs perform implicit reasoning, they are relying on shortcuts rather than engaging in true reasoning.

Furthermore, we find that the premise order does not significantly affect the model trained on an unfixed pattern. This further validates our hypothesis that such a language model relies on shortcuts for reasoning, as there is no difference in reasoning through shortcuts like chaining the numbers directly, whether in forward order (e.g., "c=6+2-3+4") or reverse order (e.g., "c=4-3+6+2").

[6]We also provide mechanistic analysis of "Variable-as-Subtrahend" in Appendix G.

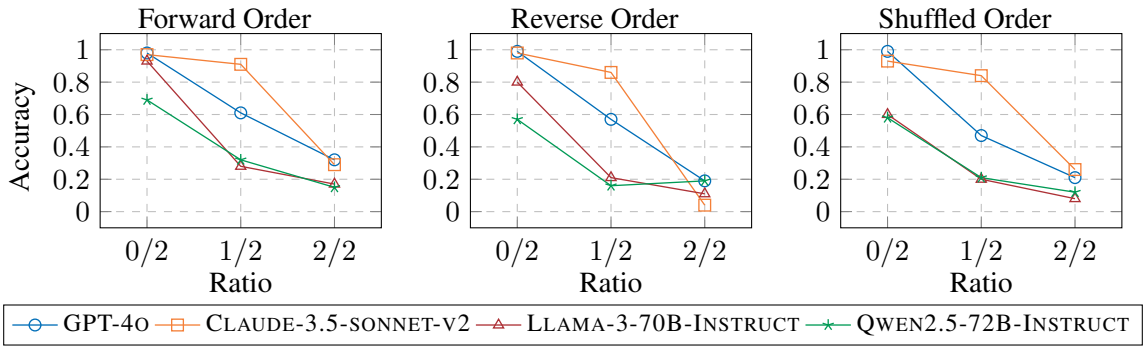


Figure 7: Performance comparison on 3-step problems with increasing numbers of equations containing a variable as the subtrahend. The problems in all the figures are the same, except for the order of premises. In a 3-step problem, at most two equations can have a variable as the subtrahend.

6 How Do LLMs Perform Multi-step Implicit Reasoning?

In the previous section, we found that GPT-2 is unable to perform implicit reasoning when there is no fixed pattern to learn during training. Does this phenomenon also apply to current SoTA LLMs, given that their training data is not always presented in a fixed order? *Do these models reason step-by-step, or rely on shortcuts to solve the problem?*

Setup. We conduct zero-shot experiments using both open-source and closed-source models, including GPT-4o-2024-08-06 (OpenAI, 2024a), Claude-3.5-sonnet-20241022-v2 (Anthropic, 2024), Llama-3-70B-Instruct (AI@Meta, 2024), and Qwen2.5-72B-Instruct (Qwen-Team, 2024). We instruct the model to provide answers directly with the temperature set to 0. To ensure the consistency and fairness of our evaluation: 1) We retain the original data generation method but restrict instances to those with intermediate computation results between 0 and 22, thus eliminating the impact of mod23 on accuracy. 2) We focus on 3-step problems, as implicit reasoning for 4-step problems proves too challenging for current LLMs, with low performance that undermines the reliability of our experiments. 3) To reduce randomness, we generate 100 problems for each ratio of equations containing a variable as the subtrahend. For each question, we evaluate it with three premise orders, i.e., forward order, reverse order, and shuffled order. 4) The accuracy is computed only in cases where the model does not output in CoT format. More details of the experimental setups are in Appendix H.

6.1 Result

Figure 7 shows the accuracy of LLMs on problems with different ratios of equations containing a variable as the subtrahend. We find: 1) As the proportion of expressions with a variable as the subtrahend increases, the accuracy of the LLMs tends to decrease drastically. The accuracy of GPT-4o even drops from nearly 100% to approximately 30% regardless of premise order. 2) All the models fail to do 3-step problems containing two equations with a variable as the subtrahend, and open-source LLMs still lag behind closed-source LLMs in implicit reasoning. 3) Compared to the influence of variables as the subtrahends, the impact of premise order is not that significant, which aligns with our models trained on unfixed premise order.

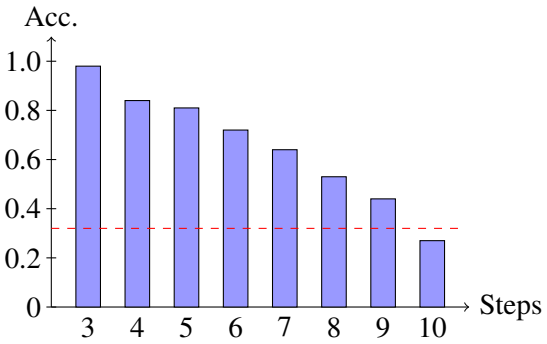


Figure 8: The accuracy of GPT-4o on problems with different step counts. The premise is in the forward order without any subtrahend being a variable. The red dashed line represents the accuracy of the same model on 3-step problems, where there are two equations with "Variable-as-Subtrahend".

We further plot the accuracy of GPT-4o on problems stated in the forward order without any subtrahend being a variable but with different steps of calculations. From Figure 8, we observe that though the accuracy of GPT-4o decreases gradually, the accuracy on 9-step problems even surpasses that on 3-step problems containing two equations with a variable as the subtrahend in Figure 7.

To sum up, the findings suggest that **LLMs**

likely rely on shortcuts for implicit reasoning rather than performing step-by-step reasoning, which aligns with our observations in the GPT-2 model. To speak further, while current LLMs can perform implicit reasoning within a fixed pattern and for a limited number of steps, they cannot generalize beyond these constraints.

7 Conclusion

In this paper, we investigate the implicit reasoning mechanism to uncover why advanced reasoning capabilities fail to emerge in the implicit reasoning style. We find that language models rely on shortcuts for implicit reasoning, and these shortcuts only work when the training data aligns with a specific pattern that supports directly chaining numbers. As a result, language models struggle with the "Variable as Subtrahend Plight," which requires true reasoning capabilities, such as variable tracking and step-by-step computation, where shortcuts are no longer effective. Experiments with current SoTA LLMs further validate our findings.

We hope this work deepens the understanding of implicit reasoning limitations in LMs and sparks future research to address LMs' key challenges in implicit multi-step reasoning.

Limitations

In this paper, we explore the mechanism of the language models performing multi-step implicit reasoning on synthetic arithmetic problems. To avoid large number operations and decimal operations, we only experiment with two fundamental arithmetic operators. This limitation suggests that future work could expand the scope to include a broader range of mathematical operators. Besides, we only focus on arithmetic reasoning due to the reasons elaborated in Section 1. We leave reasoning beyond arithmetic problems, e.g., commonsense reasoning, for further exploration in future research.

Another limitation lies in the possibility of inflated performance when evaluating LLMs. As their training methodologies and datasets remain proprietary, it is unclear whether these models were exposed to synthetic computational tasks similar to those explored in our study.

Acknowledgments

We are grateful for the constructive comments from the anonymous reviewers. We would also like to thank Jiangjie Chen from Fudan University for valuable discussions and comments on this project. This work is supported by the Chinese NSF Major Research Plan (No.92270121).

References

AI@Meta. 2024. Llama 3 model card.

Zeyuan Allen-Zhu and Yuanzhi Li. 2024. Physics of language models: Part 3.2, knowledge manipulation. *Preprint*, arXiv:2309.14402.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku.

Eden Biran, Daniela Gottesman, Sohee Yang, Mor Geva, and Amir Globerson. 2024. Hopping too late: Exploring the limitations of large language models on multi-hop queries. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14113–14130, Miami, Florida, USA. Association for Computational Linguistics.

Junhao Chen, Shengding Hu, Zhiyuan Liu, and Maosong Sun. 2024. States hidden in hidden states: Llms emerge discrete state representations implicitly. *Preprint*, arXiv:2407.11421.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. 2023. A toy model of universality: Reverse engineering how networks learn group operations. In *International Conference on Machine Learning*, pages 6243–6267. PMLR.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Yuntian Deng, Yejin Choi, and Stuart Shieber. 2024. From explicit cot to implicit cot: Learning to internalize cot step by step. *Preprint*, arXiv:2405.14838.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. 2023. Implicit chain of thought reasoning via knowledge distillation. *Preprint*, arXiv:2311.01460.

Yanai Elazar, Nora Kassner, Shauli Ravfogel, Amir Feder, Abhilasha Ravichander, Marius Mosbach, Yonatan Belinkov, Hinrich Schütze, and Yoav Goldberg. 2023. Measuring causal effects of data statistics on language model's 'factual' predictions. *Preprint*, arXiv:2207.14251.

Google. 2024. Gemini 2.0 flash thinking mode (gemini-2.0-flash-thinking-exp-1219).

Wes Gurnee, Theo Horsley, Zifan Carl Guo, Tara Rezaei Kheirkhah, Qinyi Sun, Will Hathaway, Neel Nanda, and Dimitris Bertsimas. 2024. Universal neurons in GPT2 language models. *Transactions on Machine Learning Research*.

Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. 2023. Finding neurons in a haystack: Case studies with sparse probing. *Transactions on Machine Learning Research*.

Michael Hanna, Ollie Liu, and Alexandre Variengien. 2023. How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model. In *Advances in Neural Information Processing Systems*, volume 36, pages 76033–76060. Curran Associates, Inc.

Peter Hase, Mohit Bansal, Been Kim, and Asma Ghandeharioun. 2023. Does localization inform editing? surprising differences in causality-based localization vs. knowledge editing in language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Tianjie Ju, Yijin Chen, Xinwei Yuan, Zhuosheng Zhang, Wei Du, Yubin Zheng, and Gongshen Liu. 2024. Investigating multi-hop factual shortcuts in knowledge editing of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8987–9001, Bangkok, Thailand. Association for Computational Linguistics.

Cheongwoong Kang and Jaesik Choi. 2023. Impact of co-occurrence on factual knowledge of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7721–7735, Singapore. Association for Computational Linguistics.

Keito Kudo, Yoichi Aoki, Tatsuki Kuribayashi, Shusaku Sone, Masaya Taniguchi, Ana Brassard, Keisuke Sakaguchi, and Kentaro Inui. 2024. Think-to-talk or talk-to-think? when llms come up with an answer in multi-step reasoning. *Preprint*, arXiv:2412.01113.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. 2023. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

MAA. 2024. American invitational mathematics examination.

Thomas McGrath, Matthew Rahtz, Janos Kramar, Vladimir Mikulik, and Shane Legg. 2023. The hydra effect: Emergent self-repair in language model computations. *Preprint*, arXiv:2307.15771.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. In *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372. Curran Associates, Inc.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. 2023. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. 2025. Arithmetic without algorithms: Language models solve math with a bag of heuristics. In *The Thirteenth International Conference on Learning Representations*.

nostalgebraist. 2020. interpreting gpt: the logit lens.

OpenAI. 2024a. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

OpenAI. 2024b. Learning to reason with llms.

Nikhil Prakash, Tamar Rott Shaham, Tal Haklay, Yonatan Belinkov, and David Bau. 2024. Fine-tuning enhances existing mechanisms: A case study on entity tracking. In *The Twelfth International Conference on Learning Representations*.

Qwen-Team. 2024. Qwen2.5: A party of foundation models.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *Preprint*, arXiv:2311.12022.

Alessandro Stolfo, Yonatan Belinkov, and Mrinmaya Sachan. 2023. A mechanistic interpretation of arithmetic reasoning in language models using causal mediation analysis. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7035–7052, Singapore. Association for Computational Linguistics.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomput.*, 568(C).

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. 2020. Investigating gender bias in language models using causal mediation analysis. In *Advances in Neural Information Processing Systems*, volume 33, pages 12388–12401. Curran Associates, Inc.

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. 2024. Grokking of implicit reasoning in transformers: A mechanistic journey to the edge of generalization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Jian Xie, Kexun Zhang, Jiangjie Chen, Siyu Yuan, Kai Zhang, Yikai Zhang, Lei Li, and Yanghua Xiao. 2024. Revealing the barriers of language agents in planning. *Preprint*, arXiv:2410.12409.

Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. 2024a. Do large language models latently perform multi-hop reasoning? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10210–10229, Bangkok, Thailand. Association for Computational Linguistics.

Sohee Yang, Nora Kassner, Elena Gribovskaya, Sebastian Riedel, and Mor Geva. 2024b. Do large language models perform latent multi-hop reasoning without exploiting shortcuts? *Preprint*, arXiv:2411.16679.

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. 2024. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process. *Preprint*, arXiv:2407.20311.

Yijiong Yu. 2024. Do llms really think step-by-step in implicit reasoning? *Preprint*, arXiv:2411.15862.

Zeping Yu and Sophia Ananiadou. 2024. Interpreting arithmetic mechanism in large language models through comparative neuron analysis. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3293–3306, Miami, Florida, USA. Association for Computational Linguistics.

Fred Zhang and Neel Nanda. 2024. Towards best practices of activation patching in language models: Metrics and methods. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*.

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. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 59932–59950. PMLR.

Tianyi Zhou, Deqing Fu, Vatsal Sharan, and Robin Jia. 2024. Pre-trained large language models use fourier features to compute addition. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

A More Details of the Data Generation Process

We provide an overview of the data generation process in Figure 9. First, for the training set, we create 25 000 distinct multi-step calculation templates for questions of each length (2 to 5 steps). For the 1-step template, we include all the possible combinations to enable the model to learn basic calculations. Then, we use the same method for the test set, but an additional filter mechanism is employed to prevent LMs from utilizing intermediate results from the training set. As shown in Figure 9, the model may directly utilize the result of v_1 from the training data to calculate v_2 by just calculating $v_2 = 4 + v_1$. Therefore, the test set retains only the templates whose preceding calculations, apart from the first step, do not overlap with those of the training set. This setting prevents LMs from memorizing intermediate results during training and recall them during testing rather than performing actual reasoning.

During generation, each of the two operators has a 50% probability, and the variable has a 50% probability of appearing before or after the operator (except for the first step). This results in 25% of the steps having the variable as the subtrahend in the original training and test set.

Since the variables in the template are sorted as v_0, v_1, ..., to prevent the model from learning the calculation order through the indices, we randomly replace them with the letters a-z. We use K different groups of variable names to instantiate each template in the training set. For the choice of K, please refer to the subsequent subsection.

A.1 Effect of the Number of Template Instantiations

In our early experiments, we find that when K equals 1, the model trained from scratch struggles to generalize effectively to problems outside the training set, even when these problems share the same template but have different variable names. We attempt to adjust the training hyperparameters, including the learning rate and weight decay; however, the situation remained unchanged. After increasing K to 2, the model successfully handles problems with the same template but different variable names, as well as those in the test set. So, we continue to use $K = 2$ in our experiment to ensure that the failure of generalization is caused by the model rather than our data.

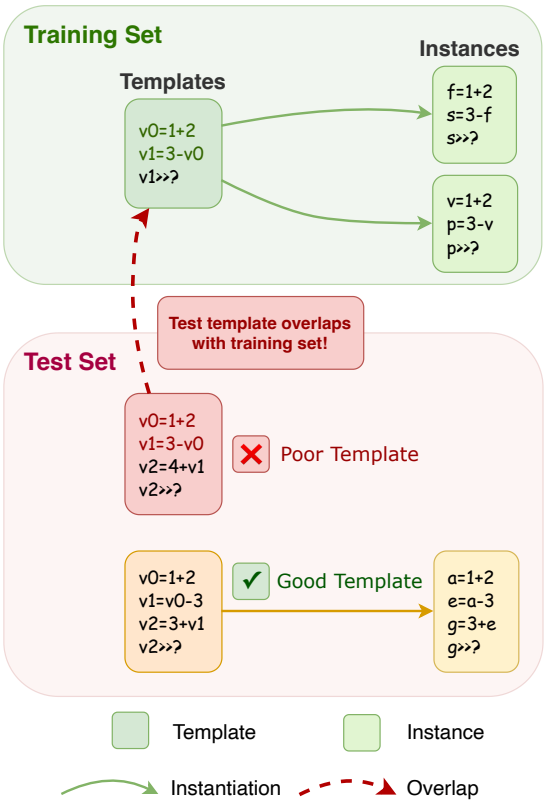


Figure 9: An overview of the data generation process.

B Choice of Activation Patching Settings

In this section, we study the choice of metrics and window sizes in the discovery of information flow.

B.1 Patching Metrics

Following the notations in Section 3, Logit denotes the output logit at the last token position, r and r' are the correct answer of the original input and corrupted input, and cl, *, pt denote the clean, corrupted and patched run separately.

In Figure 10, we compare the effect of several commonly used metrics:

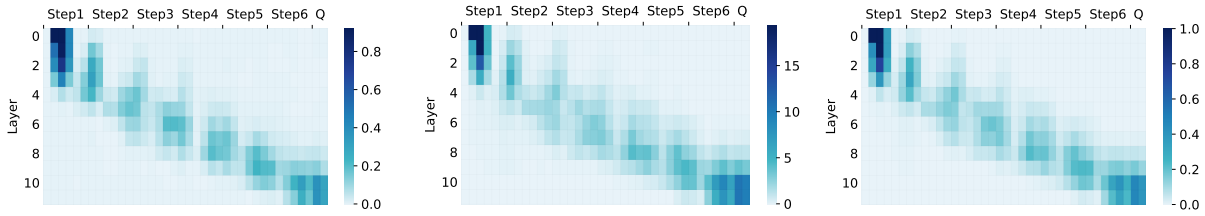
a) Logit of the clean run's ground-truth token r: $\text{Logit}_{\text{cl}}(r) - \text{Logit}_{\text{pt}}(r)$. We normalize this by $\text{Logit}_{\text{cl}}(r)$, and obtain the normalized patching effect as shown in Equation 1;

b) Logit of the corrupted run's ground truth token r': $\text{Logit}_{\text{pt}}(r') - \text{Logit}_{\text{cl}}(r')$. We do not normalize since $\text{Logit}_{\text{cl}}(r')$ can be very small, which may produce noisy localization outcomes. So we use

$$\text{PE} = \text{Logit}_{\text{pt}}(r') - \text{Logit}_{\text{cl}}(r'); \quad (2)$$

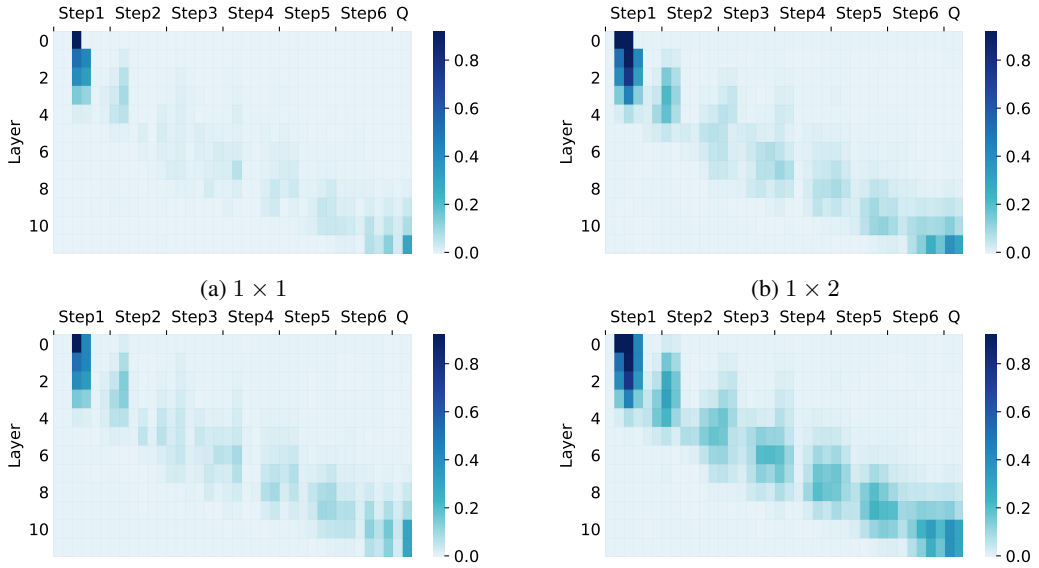
c) Logit difference: $\text{LD}(r, r') = \text{Logit}(r) - \text{Logit}(r')$. We normalize this by $\text{LD}_{\text{cl}}(r, r') - \text{LD}_{*}(r, r')$, and get

$$\text{PE} = \frac{\text{LD}_{\text{cl}}(r, r') - \text{LD}_{\text{pt}}(r, r')}{\text{LD}_{\text{cl}}(r, r') - \text{LD}_{*}(r, r')}, \quad (3)$$



(a) Logit of r (b) Logit of r' (c) Logit difference

Figure 10: Comparison of different patching metrics. (a) Logit of the clean run's ground truth token r. (b) Logit of the corrupted run's ground truth token r'. (c) Logit difference between r and r'.



(a) 1×1 (b) 1×2

(c) 2×1 (d) 2×2

Figure 11: Patching effect with different window sizes. A window size of $m \times n$ represents at each token position the representations of the region formed by the current layer and the subsequent $m - 1$ layers, along with the current token and the next $n - 1$ tokens, are copied from the corrupted forward pass.

so it typically lies in $[0, 1]$.

We find there is no significant difference in the discovery of information flow, and we use a) in our experiments for the following reasons:

1) Compared to c), a) can measure the patching effect when the ground truth tokens of the clean run and the corrupted run are the same.
2) Compared to b), the patching effect of a) can be normalized to $[0, 1]$ more stably.

B.2 Window Sizes

Following the best practices of activation patching (Zhang and Nanda, 2024), we initially employ single-layer interventions to identify crucial model components. However, as illustrated in Figure 11a, individual layer modifications produce only marginal effects, making it difficult to isolate critical hidden states. We speculate that language models may use aggregations from multiple inference pathways (McGrath et al., 2023), using a region rather than a hidden state to perform computations and restore intermediate results. Noting that the critical blocks in Figure 11a frequently exhibit rectangular patterns, we implement a 2×2 window size to capture the joint effect of these regions. Our comparison of different patching window sizes in Figure 11 reveals that different window sizes generally preserve similar information flow characteristics, and our 2×2 configuration best captures the information flow. We do not use a larger window size since the 2×2 window size is enough, and a larger window size may result in inflated localization plots.

C Information Flow Related to Operators

We present the residual stream patching plot altering the first operator in Figure 12. Similar to changing the operand, the patching effect is still pronounced at the end of each step, with information still propagating downward along the diagonal.

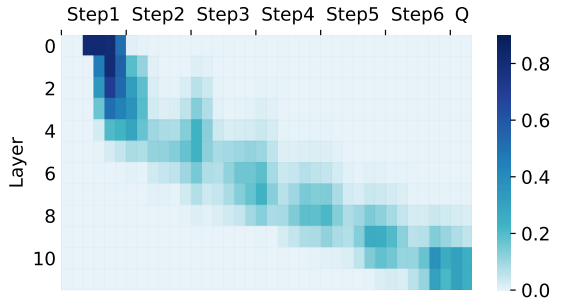


Figure 12: Activation patching on hidden states across layers and token positions when changing the first operator in the problems.

D Implementation Details of Masking Information from Previous Steps

In this experiment, we modify the Transformer model to restrict the attention so that each token can only attend to itself and the preceding $window\_size - 1$ tokens. This is achieved by applying a sliding window mask, as illustrated in Algorithm 1. Specifically, we create an attention mask, where the parts we want to focus on are set to 0, and the remaining positions are set to $-\infty$. The attention mask is then added to the attention score and passed through the softmax function. This procedure ensures the positions outside of the attention window are assigned zero attention after the softmax operation, effectively preventing the model from considering information from these positions. Through this modification, we can further confirm whether the model only utilizes the information of the current step and the intermediate result from the previous step.

Algorithm 1 Creating Sliding Window Mask

Input: $seq\_length, window\_size$
Output: $mask$

Initialize $mask$ as a $seq\_length \times seq\_length$ matrix
for $i \leftarrow 0$ to $seq\_length - 1$ **do**
  for $j \leftarrow 0$ to $seq\_length - 1$ **do**
    if $j < \max(0, i - window\_size + 1)$ **or** $j > i$ **then**
      $mask[i][j] \leftarrow -\infty$
    else
      $mask[i][j] \leftarrow 0$
    end if
  end for
end for
return $mask$

E More Details of the Training Premise Pattern

In order to enable the model to learn to reason when the order of the premises is not fixed, the training data needs to contain patterns with different premise orders. To this end, in our early experiment, we have tried several data configurations on original GPT-2-Medium to select the best one, and see whether the failure stems from the insufficiency of premise pattern. Specifically, we gradually expand the dataset by controlling the upper limit of the patterns that can be added to the training data for each template ($\times m$ indicates that at most m orders for each template will be added to the training data). If the total number of premise orders for a certain template is less than or equal to m, then all order combinations will be added to the dataset; if the total number of orders is greater than m, then for each template, m randomly selected orders of this template will be added to the dataset. We provide the sample size for each dataset in Table 3. As shown in Figure 13, including more patterns does not necessarily improve performance. When all the patterns are added, the loss of the model on the test set basically cannot decrease, and overfitting occurs rapidly.

Dataset	$\times$1	$\times$5	$\times$10	All
Size	202K	850K	1.4M	7.6M

Table 3: Size of different datasets. We control the upper limit of patterns that can be added to the dataset for each template. For example, $\times 5$ indicates that at most 5 orders for each template will be added to the training data.

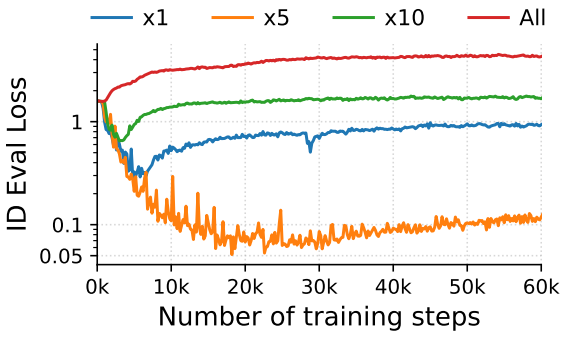


Figure 13: Loss of the model on the ID test set during training when trained with different data configurations. The y-axis is plotted on a logarithmic scale.

For the dataset including all the premise orders for every template, we also try another approach that upsample problems with fewer than 5 steps to

the same proportion as that in the $\times 5$ dataset. Although the minimum value of the evaluation loss is slightly lower, the loss increases rapidly after reaching the minimum value. We provide the accuracy of both models in Table 4. We find that both models cannot escape "Variable as Subtrahend Plight". Considering training and data efficiency, unless otherwise specified, we set the upper limit of the number of patterns for each step to 5 in all of our experiments.

Order	#Variable Being Subtrahend				
	0/4	1/4	2/4	3/4	4/4
$\times 5$					
Forward	1.00	0.90	0.90	0.30	0.05
Reverse	1.00	0.97	0.98	0.50	0.08
Random	1.00	0.83	0.89	0.45	0.23
All (Upsample)					
Forward	1.00	0.87	0.85	0.55	0.08
Reverse	1.00	0.97	0.95	0.64	0.06
Random	1.00	0.86	0.92	0.54	0.16

Table 4: Accuracy of the models with different training dataset on 5-step problems.

F Extended Experiments on Increased Data Volume and Different Models

F.1 Model Size and Initialization

When training from scratch, we also test larger models, i.e., GPT2-RoPE-medium, by increasing the number of layers from 12 to 24, but the accuracy does not improve. We find that our GPT2-RoPE model initiated from a pre-trained GPT-2's weight may alleviate overfitting and have a higher performance, but we only observe this phenomenon on GPT2-RoPE-Medium. As shown in Table 5, despite the increased accuracy, the model still fails when almost all the variables are subtrahends. In addition, investigating model initialization is not the main focus of our paper.

Order	#Variable Being Subtrahend				
	0/4	1/4	2/4	3/4	4/4
GPT2-RoPE-Medium-Pretrained					
Forward	1.00	0.98	0.99	0.97	0.05
Reverse	1.00	0.99	0.84	0.06	0.02
Random	1.00	0.90	0.83	0.35	0.08

Table 5: Accuracy of the model on 5-step questions with different numbers of variables being subtrahends.

Since training from a pre-trained model is often better than training from scratch, in order to better control the experimental variables and illustrate the impact of the model size on the experimental results, we use the original pre-trained GPT-2 series to explore the influence of the model size. In Figure 14, we present the loss curves on the ID test set for different model sizes, ranging from GPT2-Small (124M) to GPT2-XL (1.5B). We find that increasing the model size from Small to Medium can lead to improvements. However, after reaching a certain size (Medium), further increasing the model size does not yield additional gains.

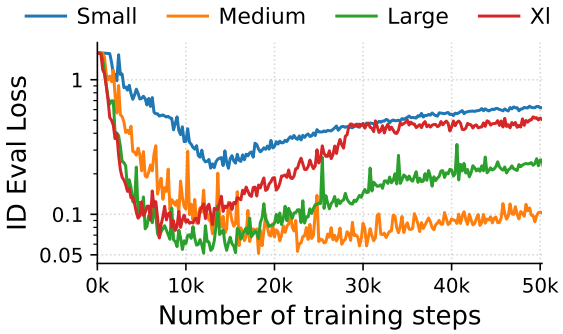


Figure 14: Loss of the model on the ID test set during training when trained with different model sizes. The y-axis is plotted on a logarithmic scale.

F.2 Model Architecture

In addition to the GPT-2 models (with or without RoPE), we also test other model architecture such as Qwen2.5. Since the performance of the pre-trained models is better than those trained from scratch, we initiate from the pre-trained weight. As shown in Table 6, we find that the model still does not escape "Variable as Subtrahend Plight".

Order	#Variable Being Subtrahend				
	0/4	1/4	2/4	3/4	4/4
Qwen2.5-1.5B-Base					
Forward	1.00	0.98	0.79	0.62	0.40
Reverse	0.99	0.97	0.98	0.90	0.69
Random	1.00	0.89	0.81	0.74	0.61

Table 6: Accuracy of the models with different architecture on 5-step problems.

F.3 Data Volume

Another potential explanation for the poor generalization could be the limited number of templates used during training. To investigate, we expand the dataset to include 50000 different templates

for each step (except for the single-step question). Since the performance of the GPT2-RoPE model used in Table 5 is better than the one used in Table 1, we continue to train from the GPT2-RoPE-Medium model initiated from a pre-trained GPT-2's weight in this experiment. As shown in Table 7, this expanded experiment yields similar results, with models still failing to generalize to problems in which most variables are subtrahends. Furthermore, we scale up the templates for 5-step problems tenfold (simultaneously upsample instances of other step counts to maintain the proportion of different step counts within the dataset), so the training dataset comprises 500K different 5-step templates, hoping the model will thoroughly learn to solve 5-step problems. In addition to the accuracy on 5-step problems in Table 7, we also visualize the model's performance on 6-step problems in Figure 15. Although continuing to scale up the data can slightly boost the model's performance within the sequence lengths seen during training, data scaling does not address the core reasoning flaw that the model cannot genuinely track variables and perform step-by-step calculations, causing it to fall into the "Variable as Subtrahend Plight".

Order	#Variable Being Subtrahend				
	0/4	1/4	2/4	3/4	4/4
50000 Templates					
Forward	1.00	0.99	0.97	0.89	0.25
Reverse	1.00	1.00	0.96	0.85	0.24
Random	1.00	0.97	0.92	0.70	0.26
50000 Templates w/ further expansion					
Forward	1.00	1.00	1.00	0.92	0.47
Reverse	1.00	1.00	0.98	0.90	0.44
Random	1.00	1.00	0.95	0.81	0.44

Table 7: Accuracy of the models with different training data volume on 5-step problems.

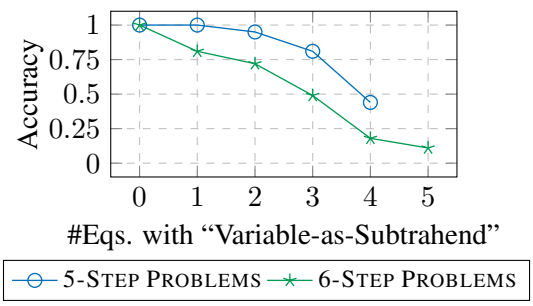


Figure 15: Test accuracies with increasing number of equations containing a variable as the subtrahend. To test genuine reasoning abilities, the premise order used during testing is random.

G Mechanistic Insights into "Variable as Subtrahend Plight"

Due to the low accuracy of the models trained from scratch, we use the GPT2-RoPE-Medium-Pretrained model from Section F.1 instead for analysis. Since the model can not fully learn to do implicit reasoning on problems requiring more than 3 steps of reasoning, we first restrict our analysis to 3-step problems. We use 1×1 patching for this model, since 1×1 patching has already had a noticeable impact.

To see why the model fails to handle equations with variables being the subtrahends, we begin our mechanistic exploration by investigating the impact of the position of the variables. Specifically, we analyze four distinct operator-variable combinations: "number + variable", "variable + number", "variable − number" and "number − variable". As shown in Figure 16, the first three graphs exhibit similar patterns, with the exception of the fourth graph, which shows some differences. We can see that in the first three graphs, the darker-colored areas are exclusively distributed in the output and numerical tokens, which means that the information in the remaining positions has no effect on the output. This phenomenon holds for all premise orders (Figure 17), since premise order does not disturb implicit reasoning through chaining the numbers directly. In contrast, in Figure 16d, we find that the dark color appears on the variable token (i.e., v_0), which means the model needs the variable value at the subtrahend position to handle subsequent calculations. We also provide the patching plot on 4-step problems in Figure 18, where a clear difference can also be observed.

These mechanistic findings show that LMs chain the numbers directly when there is no variable as the subtrahend, and explain why the premise order does not significantly affect accuracy, which validates our previous analysis in Section 5 that the model relies on shortcuts to solve the problems.

H More Details of the Experimental Setup in Section 6

In our preliminary tests, GPT-4o achieved less than 35% accuracy on 4-step problems containing only one variable as the subtrahend, while other open-source models performed only slightly above random guessing. Thus, we only study 3-step problems to ensure meaningful evaluation and better show the decreasing trend of the accuracy. Since

the first step of the operation is between numbers, there are at most two equations containing a variable as the subtrahend in 3-step problems.

As the random premise order may still contain the forward order and the reverse order, we specify a fixed shuffled order instead. Specifically, we rearrange the original premise ([step1, step2, step3]) to [step3, step1, step2]. The second step is delayed until the end, so the model can only link all the steps together at the last of the problem.

To prevent generic LLMs from using CoT reasoning to answer the question, we carefully craft the prompt to instruct the model to directly output the answer. An example of the prompt used for instructing generic LLMs to think without extra tokens in our task is shown below.

For Qwen and Llama, we use

```
a = 4 + 14
c = a - 12
s = 6 - c
What is the value of s? Please answer
directly with "s = xx".
```

and for GPT-4o and Claude, we use

```
a = 4 + 14
c = a - 12
s = 6 - c
What is the value of s? You must answer
directly. Only output the final result.
Begin your answer with "s = xx".
```

to prevent the model from outputting CoT process.

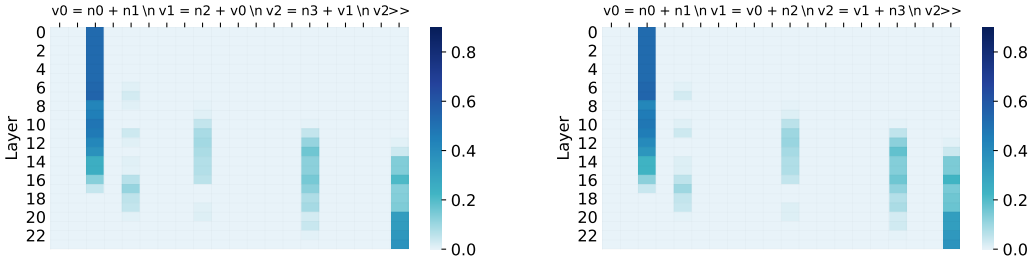
We also test questions in the form of natural language, and reach the same conclusion as shown in Table 8. This indicates that our findings are unrelated to the form of the description.

Order	#Variable Being Subtrahend		
	0/2	**1/2**	**2/2**
GPT-4o			
Forward	0.94	0.47	0.28
Reverse	0.93	0.33	0.21
Shuffled	0.88	0.39	0.15
Claude-3.5-sonnet-v2			
Forward	0.98	0.79	0.35
Reverse	0.91	0.67	0.05
Shuffled	0.85	0.64	0.20

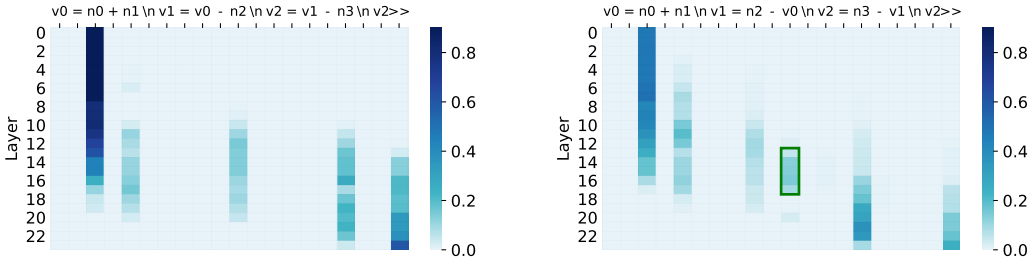
Table 8: Performance comparison on 3-step problems in the form of natural language. The problems in each column are the same, except for the premise order.

An example of the natural language form question is shown below. Here, we convert the equations in the original prompt into natural language descriptions resembling grade school math problems.

```
A's number of apples equals 4 plus 14.
C's number of apples equals A's number
of apples minus 12.
S's number of apples equals 6 minus C's
number of apples.
How many apples does S have? Only output
 the final result. Do not output
intermediate results.
```

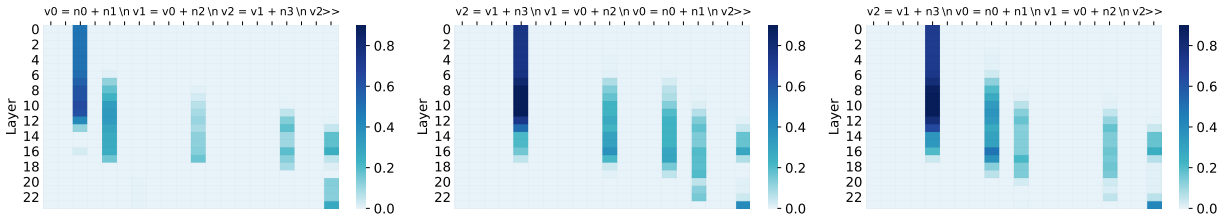


(a) number + variable (b) variable + number



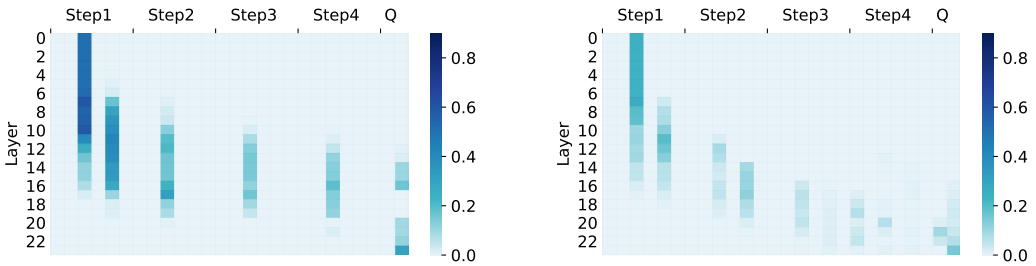
(c) variable − number (d) number − variable

Figure 16: Patching effect with different combination of the operator and the position of the variable when changing the first number in the problem.



(a) Forward order (b) Reverse order (c) Shuffled order

Figure 17: Patching effect of different premise order averaged on the same set of problems when changing the first number in the problem.



(a) No variables in the problem are subtrahends (b) Only the variable in the second step is subtrahend

Figure 18: Patching effect on 4-step problems when changing the first number. Only the second steps of the problems are different.