Language models can learn implicit multi-hop reasoning, but only if they have lots of training data

Yuekun Yao
Saarland University
ykyao@coli.uni-saarland.de

Yupei Du
Utrecht University
y.du@uu.nl

Dawei Zhu
Saarland University
dzhu@lsv.uni-saarland.de

Michael Hahn[*]
Saarland University
mhahn@lst.uni-saarland.de

Alexander Koller[*]
Saarland University
koller@coli.uni-saarland.de

Abstract

Implicit reasoning is the ability of a language model to solve multi-hop reasoning tasks in a single forward pass, without chain of thought. We investigate this capability using GPT2-style language models trained from scratch on controlled k-hop reasoning datasets ($k = 2, 3, 4$). We show that while such models can indeed learn implicit k-hop reasoning, the required training data grows exponentially in k, and the required number of transformer layers grows linearly in k. We offer a theoretical explanation for why this depth growth is necessary. We further find that the data requirement can be mitigated, but not eliminated, through curriculum learning.

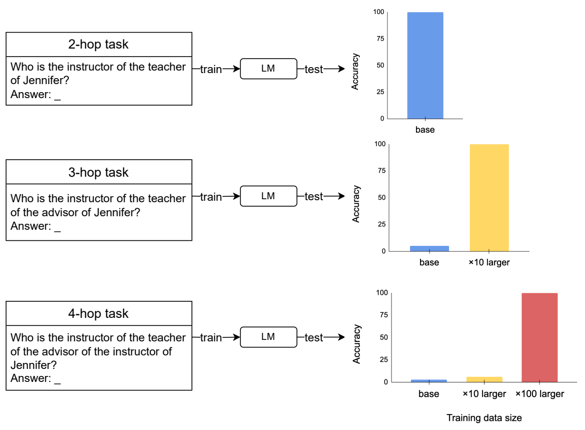


Figure 1: Example illustrating our finding: An LM can be trained to perform implicit k-hop reasoning, but requires a large increase in training data as k grows.

1 Introduction

Large language models (Brown et al., 2020; Achiam et al., 2023) have demonstrated strong capabilities in complex reasoning tasks (Jaech et al., 2024; Guo et al., 2025). With chain-of-thought methods (Wei et al., 2023; Nye et al., 2021), language models (LMs) learn to explicitly generate the intermediate steps of the given problem before generating the final answer. However, such methods incur long inference time (Chen et al., 2024b) and require costly annotations (Nye et al., 2021; Zelikman et al., 2022). This raises the question: *Can language models learn to reason effectively without explicit chain-of-thoughts, i.e., through implicit reasoning?*

There has been some research exploring implicit reasoning abilities of language models (Yang et al., 2024a; Biran et al., 2024; Wang et al., 2024). Such studies design their task in a two-hop question answering format, where the model is assumed to know individual facts like *The father of A is B* and *The teacher of B is C*, and then asked questions like *Who is the teacher of the father of A?*. Findings from these works suggest that LMs can learn implicit reasoning by combining individual factual knowledge. However, their reasoning tasks are limited to questions that can be solved with two intermediate steps (i.e. 2-hop), leaving more difficult k-hop ($k > 2$) reasoning questions alone. Hence, it remains unclear whether language models can learn to perform such k-hop reasoning or not.

In this paper, we study the capacity of language models to learn k-hop reasoning tasks, where $k = 2, 3, 4$. By training a randomly initialized GPT2-style transformer (Vaswani et al., 2017; Radford et al., 2019) on knowledge (e.g. *Jennifer 's instructor is Robert*) and knowledge-based questions (e.g. *Who is the instructor of the instructor of Jennifer?*), we study if such language models learn to generalize to questions that require novel combinations of learned facts.

Our study addresses three research questions.

- First, *can LMs learn implicit k-hop reasoning, and if so, under what conditions?* Our findings suggest that LMs can indeed learn implicit k-hop reasoning, but doing so requires exponentially increasing data budgets as k grows (see Figure 1), primarily due to the

[*] Joint senior authors.

explosion in the search space of fact combinations.

- Second, we investigate *how models perform k-hop reasoning internally* through mechanistic interpretability experiments. Our analysis reveals that models trained with sufficient data systematically derive intermediate hop entities in a layer-wise manner, progressing from shallow layers to deeper layers in a step-by-step fashion, consistent with Biran et al. (2024); Wang et al. (2024). We further show a theoretical lower bound (Theorem 5.1) suggesting that such a mechanism, with the required depth growing with k, may be unavoidable for the transformer architecture.

- Third, motivated by the substantial data requirements for k-hop reasoning, we ask: *How can we reduce the data budget for k-hop reasoning?* We explore the use of easier (m-hop, $m < k$) tasks as auxiliary training signals. Our findings show that curriculum learning (Elman, 1993; Bengio et al., 2009), which introduces tasks in a progressively harder order, significantly reduces the required training data, while simply mixing m-hop tasks with k-hop tasks provides only modest gains.

Bringing our findings together, we answer the broader question *Can language models learn implicit reasoning?* with a "yes, but" response. Language models can solve k-hop reasoning; however, this capability comes at the cost of an exponential increase in training data and at least linear growth in model depth as k increases. Curriculum learning serves as a significantly effective mitigation strategy to reduce the training data requirement, but the data growth issue still persists.

The code and datasets are available online [1].

2 Related work

Implicit reasoning. Many works have shown the power of explicit reasoning ability of language models (Wei et al., 2023; Saparov and He, 2022; Jaech et al., 2024). However, such powerful models, even after heavy pretraining (Achiam et al., 2023), generally come with negative results on implicit reasoning tasks (Press et al., 2023; Dziri et al., 2023). Relevant studies can mainly be categorized into two groups according to the evaluation task: knowledge-based reasoning (Kassner et al., 2020; Press et al., 2023; Yang et al., 2024b), and mathematical reasoning (Nanda et al., 2023; Stolfo et al., 2023). In this paper, we study the former task, and we show that GPT2-style language models, are indeed capable of multi-hop reasoning in the cost of training data requirements.

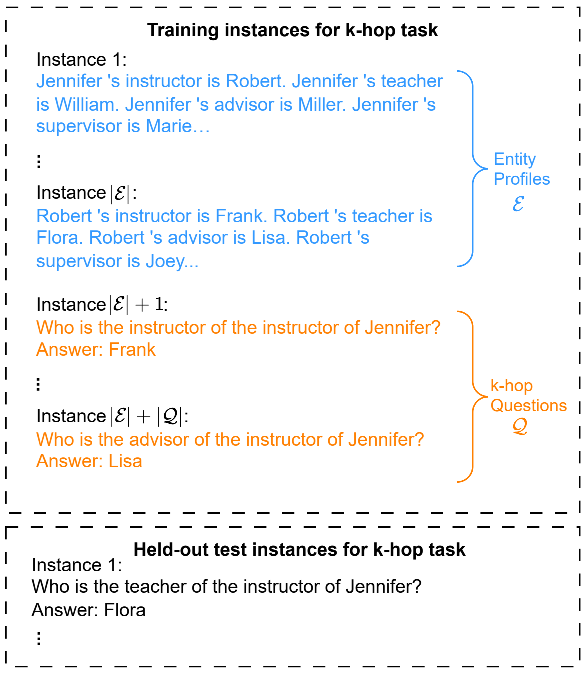
Most previous work studies knowledge-based reasoning with existing large language models (Yang et al., 2024b; Biran et al., 2024; Press et al., 2023), where language models are assumed to gain single-hop knowledge through pretraining and evaluated on multi-hop tasks. Our work instead trains language models on synthetic datasets, which allows us to accurately attribute the model behavior to particular aspects like data and models. Wang et al. (2024) also train a transformer on synthetic datasets to evaluate 2-hop reasoning. By contrast, we investigate this question across increasingly complex tasks (e.g. $2, 3, 4$-hop), and we shed light on possible methods that can help under such challenging cases.

Memorization and generalization. To train a language model to fit a training set, the model could either memorize all training instances (i.e. overfitting), or develop a generalizable solution that solves the test set. Previous work studies this in terms of grokking phenomenon (Power et al., 2022; Murty et al., 2023). Their findings suggest that both memorized and generalizable solutions exist as neural circuits in the learning process, and increasing training set size encourages the efficient one (i.e. generalizable solution) through weight decay (Nanda et al., 2023; Varma et al., 2023; Zhu et al., 2024). Compared to these work, our study suggests that training data size needs to exponentially grow according to the task difficulty, which provides a possible explanation for the failure of LLMs on complex implicit reasoning tasks.

3 Dataset

We introduce a k-hop reasoning dataset we created to train and evaluate LMs in this section. We focus on knowledge-based multi-hop reasoning, where generating the correct answer requires combining multiple known facts. Following previous work (Wang et al., 2024; Allen-Zhu and Li, 2024), we generate datasets according to synthetic knowledge, which allows better control of the task difficulty and attribution of model behaviors.

[1] github.com/ykyaol7/lm_implicit_multihop_reasoning



Training instances for k-hop task

Instance 1:
Jennifer 's instructor is Robert. Jennifer 's teacher is William. Jennifer 's advisor is Miller. Jennifer 's supervisor is Marie…

⋮

Instance $|\mathcal{E}|$:
Robert 's instructor is Frank. Robert 's teacher is Flora. Robert 's advisor is Lisa. Robert 's supervisor is Joey...

Entity Profiles $\mathcal{E}$

Instance $|\mathcal{E}|+1$:
Who is the instructor of the instructor of Jennifer?
Answer: Frank

⋮

Instance $|\mathcal{E}|+|\mathcal{Q}|$:
Who is the advisor of the instructor of Jennifer?
Answer: Lisa

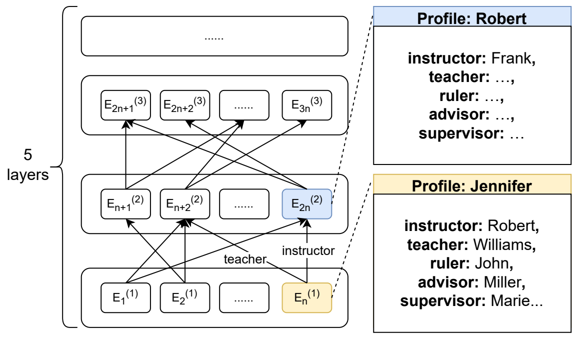
k-hop Questions $\mathcal{Q}$

Held-out test instances for k-hop task

Instance 1:
Who is the teacher of the instructor of Jennifer?
Answer: Flora

⋮

Figure 2: Example of our training and test dataset. Here, we use 2-hop task as an example.



Profile: Robert

instructor: Frank,
teacher: ...,
ruler: ...,
advisor: ...,
supervisor: ...

Profile: Jennifer

instructor: Robert,
teacher: Williams,
ruler: John,
advisor: Miller,
supervisor: Marie...

Figure 3: Profile sampling process. We always use 5 layers, and hence $n = |\mathcal{E}|/5$ (e.g. 100 for k-hop$_{large}$).

3.1 Task description

Definitions. The knowledge-based reasoning task includes two main aspects: facts and queries. Following prior definitions (Yang et al., 2024a; Wang et al., 2024), we represent a fact as a triple (e, r, e'), where e is the subject entity, r is a relation, and e' is the object entity. Each relation r acts as a function mapping a subject to an object: $r(e) \to e'$.

A k-hop query corresponds to the composition of k such functions, formalized as $r_k(r_{k-1}(\dots r_1(e)))$. Answering this query requires reasoning over a chain of k facts: (e_1, r_1, e'_1), (e'_1, r_2, e'_2), $\dots$, (e'_{k-1}, r_k, e'_k). The intermediate entities $e'_1, e'_2, \dots, e'_{k-1}$ are referred to as *bridge entities*. In a k-hop query, we refer to the components (e'_1, r_1), (e'_2, r_2), and so on as the 1-hop, 2-hop, and subsequent hops, respectively. We thus call e'_1 the 1-hop entity, and r_1 the 1-hop relation. While prior work has mostly focused on 2-hop queries involving a single bridge entity, we construct datasets for $k \in \{2, 3, 4\}$ to assess models' ability to handle increasingly complex reasoning chains.

Dataset format. We create one dataset for each $k \in \{2, 3, 4\}$ task. Our dataset includes two components: (1) entity profiles encoding known facts, and (2) reasoning questions that query compositions of facts in natural language (see Figure 2).

- An entity profile encodes all possible facts for a particular entity where the entity serves as the subject entity (e.g. *Jennifer 's instructor is Robert, Jennifer 's teacher is William...*).

- The prompt for our reasoning question is as simple as "*Who is the teacher of the instructor of Jennifer? \n Answer:* ", where *instructor, teacher* refer to relations and *Jennifer* refers to the queried entity.

We introduce details for generating profiles and questions in Section 3.2. To ensure the model has access to all entity profiles, the training set includes all possible profiles together with randomly selected reasoning questions, and we use the held-out reasoning questions as the test set.

We construct two dataset variants by varying the number of entities ($|\mathcal{E}|$) and relations ($|\mathcal{R}|$): a larger dataset with $|\mathcal{E}|= 500$, $|\mathcal{R}|= 20$ (denoted k-hop$_{large}$), and a smaller one with $|\mathcal{E}|= 250$, $|\mathcal{R}|= 10$ (denoted k-hop$_{small}$).

3.2 Data generation

Profile sampling. We use the same set of entity profiles across $k = 2, 3, 4$ tasks to ensure fair comparison. Figure 3 illustrates the process to generate profiles. We first sample $|\mathcal{E}|$ entity names (e.g. *Jennifer*) from a predefined namespace and group them into K disjoint hierarchical layers, where K is the largest $k + 1$ value. Since we consider $k < 5$, K is fixed at 5. Each entity is then linked to $|\mathcal{R}|$ randomly selected entities in the upper layer through distinct relations, with relation names reused across layers for generality. This structure guarantees that a composition of $k \in \{2, 3, 4\}$ relations starting from any entity in the bottom layer leads to a well-defined target entity. More details are provided in Appendix D.1.

Profile and question generation. Each fact (e, r, e') is mapped to a natural language sentence

Dataset		Training data budget ×1	×2	×5	×10	×20	×50	×100
k-hop*small*	2-hop	99.8						
	3-hop	5.7	12.6	99.9	100			
	4-hop	4.3	6.2	6.7	9.2	96.4	100	100
k-hop*large*	2-hop	99.9						
	3-hop	2.5	3.1	4.9	94.6	100		
	4-hop	2.0	2.6	3.1	3.7	4.0	6.3	100

Table 1: Accuracy of GPT-2 on k-hop*small* and k-hop*large* datasets with different training data budgets. Empty cells indicate that the data budget exceeds the number of possible questions.

using a simple template (e.g., *{subj}'s {relation} is {obj}*). Following previous work (Allen-Zhu and Li, 2024), all facts about a given subject entity are concatenated into a single paragraph to form that entity's profile. To construct reasoning questions, we sample entities from the bottom layer of the hierarchy (Figure 3) and recursively traverse k relations to identify the correct answer. All valid k-hop queries are generated for each source entity. For example, for 2-hop queries on k-hop*large*, we can generate up to $|\mathcal{R}|^2 \times |\mathcal{E}|/5 = 40000$ instances.

4 LMs can learn k-hop reasoning, but at a large data cost

Our first objective is to establish that language models can learn implicit k-hop reasoning, but this requires the number of training instances (i.e. k-hop reasoning questions) grows exponentially as k increases. In this section, we empirically demonstrate this by training models on our k-hop datasets with $k = 2, 3, 4$.

4.1 Experiment setup

Model. We adopt the smallest GPT-2 architecture (Radford et al., 2019) as our model. Following recent studies (Allen-Zhu and Li, 2024), we replace the original positional embeddings in GPT-2 with Rotary Position Embedding (RoPE) (Su et al., 2024). We use the GPT-2 tokenizer (Radford et al., 2019) and extend its vocabulary by adding all possible entity names from our dataset. The training objective is the causal language modeling loss calculated over all tokens in each prompt. In our main experiments, we train the model from scratch by randomly initializing all parameters. Additionally, we conduct experiments using the pretrained GPT-2 and its larger variants (see Appendix B for results).

Training. We set the training steps to 20k for all tasks except 4-hop*large*, where we extend the training to 40k steps to ensure convergence. We apply a cosine learning rate scheduler with 1k warm-up steps. Each experiment is repeated across three runs using different random seeds, and we report the average performance. Details of hyperparameters for model architecture and training are provided in Appendix E.

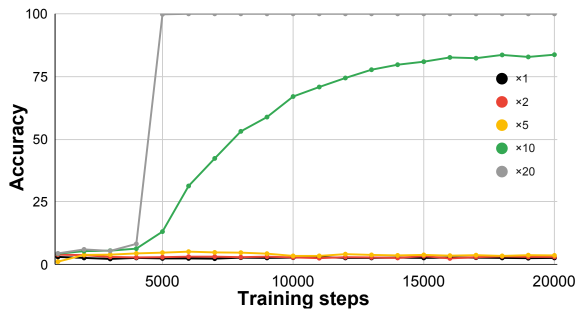
Dataset. We utilize the k-hop*small* and k-hop*large* datasets introduced in Section 3 for training and evaluation, considering $k = 2, 3, 4$. This results in six datasets in total. For the 2-hop task, we generate all possible reasoning questions and randomly sample 50% for the 2-hop*large* training set and 80% for 2-hop*small*. All entity profiles are included in the training sets. The test set consists of 3,000 instances randomly selected from the held-out questions, except for 2-hop*small*, which contains only 1,000 held-out questions. We report the details and statistics of our datasets in Appendix D.1.

For 3-hop and 4-hop tasks, we find that the same data size as the 2-hop training set results in random guessing performance. Thus, we progressively increase the training data size by defining the base training budget bg as the number of reasoning questions in the 2-hop training set. We create training sets by scaling bg with ratios from the set $\{\times 1, \times 2, \times 5, \times 10, \times 20, \times 50, \times 100\}$. For each ratio r, we randomly sample $r \times bg$ reasoning questions for training. The test set for each k-hop task always includes 3,000 instances randomly sampled held-out instances except for 2-hop*small*.

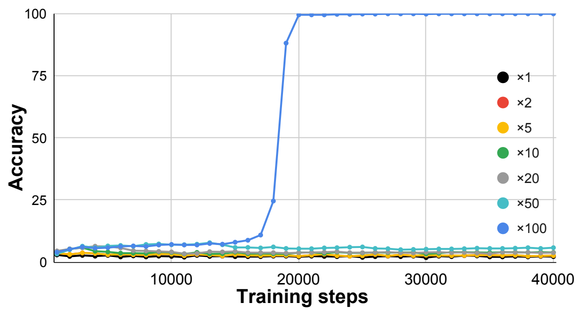
Evaluation. For each test instance, we provide the language model with the prompt up to the answer token (e.g., "*Who is the instructor of the instructor of Jennifer? \n Answer:* ") and evaluate the accuracy of the generated token against the gold answer. Greedy decoding is used for evaluation.

4.2 Results

Language models can learn k-hop reasoning. Table 1 reports the test accuracy of our models under



(a) 3-hop$_{large}$



(b) 4-hop$_{large}$

Figure 4: Model accuracy on 3-hop$_{large}$ and 4-hop$_{large}$. 3-hop$_{large}$ can only be solved when the training budget is increased by at least a factor of $\times 10$, while 4-hop$_{large}$ requires $\times 100$. Using $\times 20$ budget further encourages convergence on 3-hop$_{large}$ compared to $\times 10$ budget.

varying training data budgets. Our first observation is that GPT-2 models are capable of achieving 100% accuracy not only on 2-hop tasks but also on more complex 3-hop and 4-hop tasks, given a sufficiently large training data budget with the same k as the test set. This is a significant finding, as each entity profile appears individually in the training set without any explicit instructions on how to combine them to solve multi-hop tasks. The perfect accuracy suggests that language models can learn the underlying reasoning process based solely on input-output pairs, even without explicit rationales.

However, data requirements increase exponentially with k. We further observe that the base training data budget ($\times 1$) is insufficient for the model to effectively learn 3-hop and 4-hop tasks, as evidenced by test accuracy below 10%. As the training data budget increases, model performance improves correspondingly. We define a model as successfully learning the task if it achieves a test accuracy above 80%. On k-hop$_{small}$ datasets, a minimum budget of $\times 5$ is necessary to learn the 3-hop task, whereas the 4-hop task requires a budget of at least $\times 20$. On k-hop$_{large}$ datasets, the data budget required for the 3-hop task is $\times 10$, and for the 4-hop task, it escalates to $\times 100$. These findings suggest that the training data budget grows in an exponential manner as the value of k increases.

We also plot the test accuracy of one training run on k-hop$_{large}$ across training steps in Figure 4 (For k-hop$_{small}$ results see Appendix F.1). The plots show that a larger training budget not only results in higher accuracy but also accelerates model convergence. For instance, in Figure 4a, the $\times 20$ budget reaches 100% accuracy by step 5000, while the $\times 10$ budget only achieves 10% accuracy at the same step. This finding is also consistent with

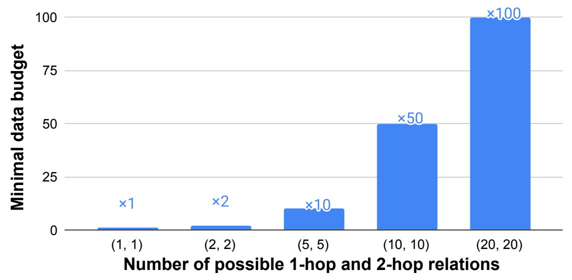


Figure 5: Case study on 4-hop$_{large}$. x-axis denotes the number of 1-hop and 2-hop relations, e.g. $(1, 1)$ denotes that 1-hop and 2-hop relations are fixed across all 4-hop questions. Fixing 1-hop and 2-hop relations reduces the required training budget to $\times 1$, while increasing them leads to rapid budget growth.

Wang et al. (2024) reported in 2-hop reasoning tasks. We extend these observations by demonstrating that the data budget becomes even more critical as the complexity of the reasoning task increases.

4.3 Why data-hungry?

Results so far highlight the substantial data requirements for k-hop tasks, but the reason for this remains unclear. Increasing the value of k leads to both an increase in the number of combined facts (i.e., k facts for each entity) and a corresponding exponential increase of the search space (i.e., $|\mathcal{R}|^k$ relation combinations per entity). Our objective here is to disentangle the effects of these two factors and identify the primary source of data inefficiency.

Setup. To investigate this question, we conduct a case study on the 4-hop$_{large}$ dataset, where we vary the number of 1-hop and 2-hop relations while holding the number of relations in the 3-hop and 4-hop positions constant. In the original dataset, each hop position can take one of $|\mathcal{R}|= 20$ possible relations. For this study, we generate new training and test sets by limiting the number of 1-hop and 2-

hop relations to values from the set $\{1, 2, 5, 10, 20\}$. For each configuration, we train GPT-2 models and determine the minimal data budget required to achieve 80% test accuracy.

Figure 5 presents the results. We observe that when the number of 1-hop and 2-hop relations is restricted to a single relation, the model can successfully learn 4-hop task using the base data budget. However, as the number of relations increases, the required data budget rapidly increases. This result suggests that the main source of data inefficiency in k-hop reasoning tasks is the exponential growth in the number of relation combinations, rather than the number of individual facts to be combined.

5 LMs reason through layer-wise lookup, incurring the cost of depth

The second objective is to understand the underlying mechanism by which language models solve the k-hop task. We first demonstrate that language models solve such tasks by layer-wise lookup of bridge entities of a k-hop query $r_k(r_{k-1}(\dots r_1(e)))$ through empirical evidence (e.g. mechanistic interpretability). Building on this finding, we then establish a theoretical lower bound, showing that the model's depth must grow with k to maintain such layer-wise lookup mechanism.

5.1 Experiment setup

We design two experiments to investigate the model's internal reasoning process: probing and causal intervention. For both experiments, we select the model trained on 4-hop$_{large}$ with a $\times 100$ budget, as it achieves strong performance.

Probing. We use probing tasks (Belinkov and Glass, 2019; Liu et al., 2019) to assess whether information about intermediate bridge entities is encoded in the hidden representations. In this setup, we freeze the model parameters and train a linear probe classifier on top of the hidden states to predict the correct entity. We train one probe classifier for each hop position, predicting the corresponding bridge entity in the query. The probe is trained across all transformer layers and all tokens in the prompt to identify where and when information about the bridge entities is encoded. We split the 4-hop$_{large}$ test set into $80/20\%$ training and evaluation sets for training the probe classifiers.

Causal intervention. While probing shows whether information about bridge entities is encoded in the hidden representations, it does not tell us whether the model actually relies on this information to generate the final answer. We further design *activation patching* (Vig et al., 2020; Meng et al., 2022) experiments to investigate it.

The core idea of activation patching is to replace the residual stream (i.e., the output of a residual layer in a transformer block) at a specific layer L_i and prompt token t_j, and measure the resulting change in the output probability of the correct answer. For convenience, we call this residual stream $res(L_i, t_j)$. In this section, we focus on the last token position in the input prompt (i.e., t_j always being the last input token, which is whitespace <*space*>), as justified in Section 5.2.

Suppose we are given a k-hop test instance and aim to measure the causal effect of $res(L_i, t_j)$, the residual stream at layer L_i and token t_j. For clarity, we define three types of runs as follows. **Clean Run:** The original forward pass of the test instance, producing the output probability of the correct answer as P_{clean}. **Corrupted Run:** A distinct k-hop instance selected to serve as the source of the patched residual stream. **Patched Run:** The modified run, where the residual stream $res(L_i, t_j)$ in the clean run is replaced with the corresponding $res(L_i, t_j)$ from the corrupted run, leaving other layers unchanged. The output probability in the patched run is denoted as P_{patched}. The causal effect of the targeted residual stream is defined as $P_{\text{clean}} - P_{\text{patched}}$, where a larger effect indicates greater reliance on the removed information. We calculate the causal effect for each layer and report the average effect across 3000 held-out instances.

The aim of our intervention experiment is to measure the effect of bridge entity information at different hop positions (e.g., 1-hop, 2-hop). Hence, we define four types of corrupted runs for each clean run: $C_{\text{1-hop}}$, $C_{\text{2-hop}}$, $C_{\text{3-hop}}$, and $C_{\text{4-hop}}$. In a $C_{\text{i-hop}}$ run, we select a corrupted instance where the gold i-hop entity differs from the clean run, while the entities of other hop positions remain unchanged. This setup allows us to measure the effect of perturbing a specific i-hop entity while keeping other bridge entities unchanged.

5.2 Results

Bridge entities are encoded in the last token position. Figure 6 presents the probing results across layers and token positions in the input prompt, e.g., *Who is the instructor of the teacher of the advisor of the instructor of* <*Entity*>*? \n Answer:*<*space*>. We report results only for tokens after the <*En-*

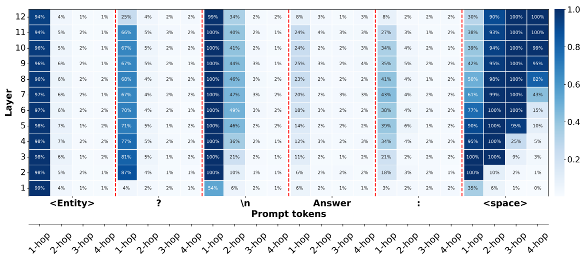


Figure 6: Probing results across tokens in the input prompt. Each token is represented by four columns, corresponding to 1-hop to 4-hop bridge entities. Note that tokens preceding <*Entity*> cannot include information about any of the entities, and are thus not shown here. Only the <*space*> token consistently encodes information about all four bridge entities, indicating that reasoning is concentrated at the last token before answer generation.

tity> token, as preceding tokens cannot contain information about the target bridge entities. Since the vocabulary size of each i-hop entity is 100, a random baseline provides 1% accuracy.

Notably, the hidden representation of the last input token encodes information about all necessary bridge entities for predicting the final answer. Instead, probe classifiers show low accuracy for other token positions, suggesting that the reasoning process likely occurs in the position immediately before generating the final answer. We confirm this by observing zero casual effects on preceding tokens with additional activation patching experiment (see Appendix G.1). We thus focus our causal intervention experiments on this <*space*> token.

Output prediction relies on bridge entity information. Figure 7 shows the causal effects across layers in our intervention experiment. For each i-hop entity, we identify specific layers that the model relies on to generate the final answer. Moreover, the model organizes the reasoning process in a layer-wise manner, with shallower layers handling lower-hop entities, and deeper layers handling higher-hop entities. This layer-wise lookup confirms that the model leverages bridge entity information to perform multi-hop reasoning, which generalizes prior observations from 2-hop tasks (Biran et al., 2024; Wang et al., 2024).

5.3 Theoretical Analysis

We have found that language models perform the k-hop task by layer-wise lookup. This suggests that transformers may need depth *linear* in the number of reasoning steps. Here, we discuss how this result relates to the in-principle expressiveness of trans-

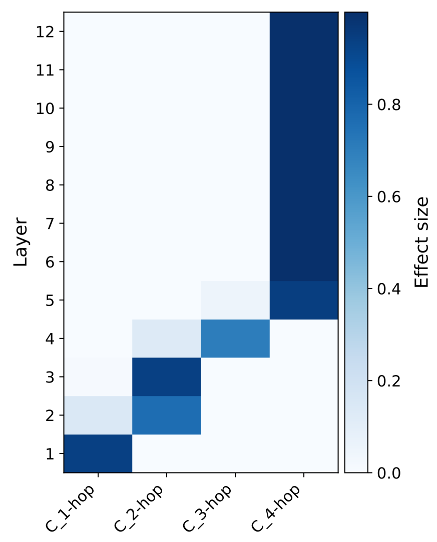


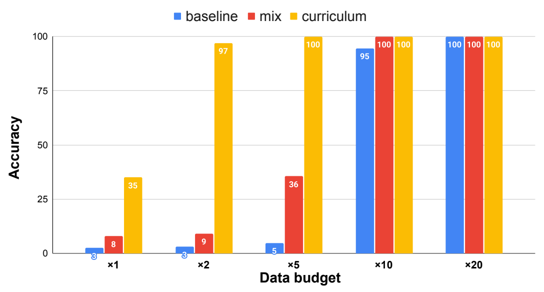
Figure 7: Causal interventions reveal a layer-wise lookup mechanism: Intervening on the 1-hop entity has the strongest effect in the 1st layer, and little effect in higher layers. Intervening on the 2-hop entity has an effect mainly in the 2nd and 3rd layers; analogously for the 3-hop and 4-hop entities. Overall, these results indicate that entities are looked up layer-by-layer.

formers. Formally, we consider a universe $\mathcal{E}$ of entities (e.g., $\{$*Jennifer*, *Frank*, $\dots\}$) and a set $\mathcal{R}$ of maps $r : \mathcal{E} \to \mathcal{E}$ (e.g., when r = *instructor*, $e = Jennifer$, then $r(e)$ denotes the instructor of Jennifer). We consider the task of mapping an input string *"Who is the r_k of the r_{k-1} $\dots r_2$ of the r_1 of e? Answer:"* (as in Figure 2) to the entity $r_k(\dots r_1(e)\dots) \in \mathcal{E}$ (e.g., *instructor(teacher(Jennifer))*). We lower-bound the number of layers needed in the case where the *attention pattern does not depend on the query e*. We consider this a reasonable special case, as there is no obvious way in which query-dependent attention would help solve the k-hop task. In this case:

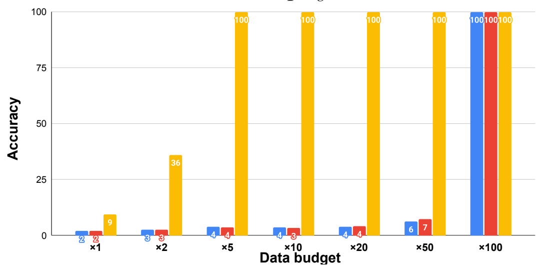
Theorem 5.1 (See Appendix A for proof)**.** *Consider a causal transformer operating in p bits of precision, with d hidden units, H heads and L layers. Assume it performs k-hop reasoning over $\mathcal{E}$ and $\mathcal{R}$ as defined above. Assume further that the attention pattern does not depend on e. If $k \leq |\mathcal{E}| - 2$, then, for some $\mathcal{R}$,*

$$L \geq \frac{k}{8pdH} \qquad (1)$$

We note that there are relation sets $\mathcal{R}$ for which shortcuts with few layers may exist, but the result shows that a linear number is needed in the worst case. This statement expresses a *width-depth tradeoff*: the product of the number of layers, bits of



(a) 3-hop$_{large}$



(b) 4-hop$_{large}$

Figure 8: Model performance on k-hop$_{large}$ datasets with mixed learning and curriculum learning. Curriculum learning enables the model to solve the 4-hop task with a $\times 5$ training budget, compared to the $\times 100$ required by both the baseline and mixed learning setups.

precision, width, and number of heads needs to grow linearly in k. In particular, within a single model (i.e., fixing d, H, and p), the analysis predicts that, as k grows, more and more layers need to be involved in the hop-by-hop retrieval, as we found empirically (Figure 7). We also note that existing results (Chen et al., 2024a) are not applicable to our k-hop task (Appendix A.2). Further empirical evidence supports our theoretical prediction (see Appendix C).

6 Curriculum learning mitigates the data requirement, but doesn't solve it

Finally, we study training strategies to improve the data budget issue. Models in Section 4 were trained solely on k-hop task, but i-hop ($i < k$) questions should also be available in realistic setups. By exploiting such easier questions as additional training data for k-hop task, we demonstrate that curriculum learning significantly mitigates the exponential growth issue, though not eliminate the increase of data budget as k increases.

6.1 Experiment setup

We use the same GPT-2 architecture as in Section 4.1 and compare two strategies: mixed learning and curriculum learning (Bengio et al., 2009).

Mixed learning. We construct the training set by combining reasoning questions from both lower-hop and k-hop tasks. For instance, the 4-hop training set contains a mix of 2-hop, 3-hop, and 4-hop questions, along with all relevant entity profiles. Lower-hop questions are generated using the same entity profiles as the target task. We vary the k-hop training budget using the same scaling factors as in Section 4.1, while keeping the amount of lower-hop data fixed (see Appendix D.2 for dataset details).

Curriculum learning. Training in curriculum learning is split into multiple stages, where each stage progressively introduces harder reasoning tasks. For a k-hop task, training proceeds in $k-1$ stages: the first stage uses only 2-hop questions, the second stage includes both 2-hop and 3-hop, and so on. We use the same lower-hop data as in the mixed learning setup and ensure that total training steps are equal across both strategies. See Appendix E.2 for training details.

Test set. To avoid shortcut solutions (e.g., where lower-hop queries appear as subcomponents of the k-hop query), we generate test sets such that such overlaps do not exist using rejection sampling. The test set size remains 3000 instances, consistent with previous experiments.

6.2 Results.

Figures 8 shows the results for k-hop$_{large}$, and the same pattern holds for k-hop$_{small}$ (see Appendix F.1 for results). We compare mixed learning, curriculum learning, and the baseline model trained only on the target k-hop dataset from Section 4.1.

Curriculum learning significantly reduces the required data budget. Notably, curriculum learning yields the most significant improvement. For example, perfect accuracy on 4-hop tasks is achieved with only a $\times 5$ budget, compared to $\times 100$ in the baseline. In contrast, simply mixing all available data provides only modest gains. This demonstrates that presenting easier reasoning tasks before harder ones is a highly effective strategy for improving data efficiency.

Curriculum learning builds circuits gradually. We attribute this effectiveness of curriculum learning to a stepwise build-up of circuits: As we show in Appendix G.2, mechanisms retrieving lower-hop entities (e.g., 1-hop) emerge in the early training stages; subsequent stages then build upon these established circuits to learn more complex reasoning tasks. While baseline models have to construct a full circuit for k-hop reasoning at once, curriculum learning enables 1-hop circuits to emerge in shallower layers in the first stage, with later stages developing circuits for 2-hop and 3-hop entities on top of these.

Curriculum learning does not completely solve the data growth issue. Despite the effectiveness of curriculum learning strategy, it does not completely eliminate the growth of data budget. For example, curriculum learning requires $\times 2$ budget for 3-hop task and $\times 5$ for 4-hop task, indicating the challenge of k-hop implicit reasoning for LMs.

7 Conclusion

Our work investigates whether language models can learn implicit multi-hop reasoning. We provide a nuanced answer through controlled k-hop reasoning datasets using GPT2-style language models. On the one hand, our findings demonstrate that language models can indeed learn k-hop reasoning through sequential lookup of intermediate bridge entities layer by layer. However, this capability comes at a cost: as k increases, the training data budget grows exponentially, and the model depth must scale linearly. Furthermore, while curriculum learning mitigates the data budget growth, it does not eliminate the growth trend. Together, we present a comprehensive view of the potential and limitations of LMs in implicit reasoning, underscoring the inherent trade-offs between task complexity, data requirements, and model depth.

8 Limitations

We limit our study to implicit reasoning tasks using synthetic datasets generated based on predefined templates. Applying the same analysis to realistic datasets is challenging due to the difficulty of collecting complex multi-hop questions (e.g. 4-hop questions) and corresponding facts. Due to computational budget constraints, we also restrict our experiments to k-hop tasks with $k < 5$.

Additionally, our experiments primarily rely on randomly initialized small language models (GPT-2 small). While we also observe that the data budget issues persist for pretrained models (e.g. pretrained GPT-2) and larger models (GPT-2 medium and large with up to 770M parameters), we do not extend our analysis to models with greater parameter sizes.

Acknowledgments

We thank Lijie Chen for fruitful discussions, and Ji-Ung Lee and Katharina Stein for their feedback on the paper.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Zeyuan Allen-Zhu and Yuanzhi Li. 2024. Physics of language models: Part 3.1, knowledge storage and extraction. In *International Conference on Machine Learning*, pages 1067–1077. PMLR.

Yonatan Belinkov and James Glass. 2019. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

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.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Lijie Chen, Binghui Peng, and Hongxun Wu. 2024a. Theoretical limitations of multi-layer transformer. *arXiv preprint arXiv:2412.02975*.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. 2024b. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*.

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

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. 2023. Faith and fate: Limits of transformers on compositionality. *Advances in Neural Information Processing Systems*, 36:70293–70332.

Jeffrey L Elman. 1993. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99.

Erwan Fagnou, Paul Caillon, Blaise Delattre, and Alexandre Allauzen. 2024. Chain and causal attention for efficient entity tracking. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13174–13188, Miami, Florida, USA. Association for Computational Linguistics.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*.

Peter M Higgins. 2017. Embedding in a finite 2-generator semigroup. *Glasgow Mathematical Journal*, 59(1):61–75.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models.

Nora Kassner, Benno Krojer, and Hinrich Schütze. 2020. Are pretrained language models symbolic reasoners over knowledge? In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 552–564, Online. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Alexander Kozachinskiy, Felipe Urrutia, Hector Jimenez, Tomasz Steifer, Germán Pizarro, Matías Fuentes, Francisco Meza, Cristian B Calderon, and Cristóbal Rojas. 2025. Strassen attention: Unlocking compositional abilities in transformers based on a new lower bound method. *arXiv preprint arXiv:2501.19215*.

Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. 2019. Linguistic knowledge and transferability of contextual representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1073–1094, Minneapolis, Minnesota. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372.

William Merrill and Ashish Sabharwal. 2023. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher Manning. 2023. Grokking of hierarchical structure in vanilla transformers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 439–448, Toronto, Canada. Association for Computational Linguistics.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. 2023. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show your work: Scratchpads for intermediate computation with language models.

A Paszke. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

Binghui Peng, Srini Narayanan, and Christos Papadimitriou. 2024. On limitations of the transformer architecture. In *First Conference on Language Modeling*.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. 2022. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, Singapore. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Abulhair Saparov and He He. 2022. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*.

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.

Lena Strobl. 2023. Average-hard attention transformers are constant-depth uniform threshold circuits. *arXiv preprint arXiv:2308.03212*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. 2023. Explaining grokking through circuit efficiency. *arXiv preprint arXiv:2309.02390*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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. *Advances in neural information processing systems*, 33:12388–12401.

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. 2024. Grokking of implicit reasoning in transformers: A mechanistic journey to the edge of generalization. *Advances in Neural Information Processing Systems*, 37:95238–95265.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

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? *arXiv 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. In *The Thirteenth International Conference on Learning Representations*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

Xuekai Zhu, Yao Fu, Bowen Zhou, and Zhouhan Lin. 2024. Critical data size of language models from a grokking perspective. *arXiv preprint arXiv:2401.10463*.

A Details for Theoretical Results

A.1 Proof of Theoretical Bound

Theorem A.1 (Restated from 5.1)**.** *Consider a causal transformer operating in p bits of precision, with d hidden units, H heads and L layers. Assume it performs k-hop reasoning over $\mathcal{E}$ and $\mathcal{R}$ as defined above. Assume further that the attention pattern does not depend on e. If $k \leq |\mathcal{E}| - 2$, then, for some $\mathcal{R}$,*

$$L \geq \frac{k}{8pdH} \tag{2}$$

Proof. Recall that the input has the form

Who is the r_k of the $r_{k-1} \dots r_2$ of the r_1 of e ?
Answer :

Our argument is based on communication complexity. We consider a communication game where Alice holds $r_1, \dots, r_k$, and Bob holds e. Due to causal masking, Alice can compute the transformer's activations on all tokens "Who is the r_k of the $\dots r_1$ of" without receiving any information from Bob. In order to compute activations on then final tokens "e ? Answer :" (and thus the prediction), Bob requires access to the outputs of attention heads on these tokens. Because the attention patterns are assumed to be independent of the query e, Alice can simple, for each head at the four tokens "e ? Answer :" provide the activations within the span known to Alice weighted with the attention weights. Thus, a total of $4HL$ such activation vectors is sufficient. Furthermore, each of these activations can be encoded with pd bits. Overall, thus, Bob can compute the output with access to only $4LpdH$ bits.

That is, there a way to compress the composed function $r_k \circ \dots \circ r_1$ into $4LpdH$ bits. Hence,

2^{4LpdH} upper-bounds the cardinality of such possible functions:

$$4LpdH \geq \log_2 |\{r_k \circ \dots \circ r_1 : r_1, \dots, r_k \in \mathcal{R}\}| \quad (3)$$

We note that, in general, the right-hand-side could be small: for instance, if $\mathcal{R}$ just contains the identity function, then the set of k-fold composed functions will also just contain the identity function (since its composition with itself again equals itself). To conclude the theorem, the remaining problem is now to show that there is a way of choosing $\mathcal{R}$ for which the right-hand-side scales with k.

We arbitrarily label the elements of $\mathcal{E}$ as $\{0, x_0, x_1, \dots, x_{n-1}\}$, and define $\mathcal{R} = \{f, g\}$ where:

$$\begin{aligned} f(0) &= 0 \\ f(x_i) &= x_{(i+1)\%n} \\ g(0) &= 0 \\ g(x_0) &= 0 \\ g(x_i) &= x_i \quad (i > 0) \end{aligned}$$

Intuitively, (1) there is a special "sink" entity 0, (2) f cyclically shuffles the non-sink entities, (3) g maps the first entity in the order to the sink entity. We now consider all words over f, g of length k where gg does not occur (recall that, by assumption, $k \leq |\mathcal{E}| - 2 = n - 1$). The number of such words is exponential in k; indeed, it is at least $\left(\frac{3}{2}\right)^k$.[2] Each such word, interpreted as a composition, generates a different transformation $\mathcal{E} \to \mathcal{E}$.[3] Indeed, to show this, we simply note that such a composition maps x_i to 0 if and only if g was applied immediately after the $i+1$-th application of f. Thus, when $k \leq |\mathcal{E}| - 2$, we have lower-bounded the right-hand-side of (3) as $k \cdot \log_2 \frac{3}{2} > \frac{k}{2}$. The theorem then follows by rearranging (3). □

A.2 Discussion

Related Work Chen et al. (2024a) prove a lower bound on L for causal transformers solving a more complicated kind of composition task, which composes functions taking *two* arguments. The input provides both (i) a sequence of functions, and (ii) a sequence of entities serving as the second argument, with the output

$$z_{l+1}(w_l, z_l(w_{l-1}, z_{l-1}(\dots z_2(w_1, i_1)))) \quad (4)$$

where $z_1, \dots, z_{l_1}$ are two-argument functions, i_1 can be viewed as an input entity (similar to the query in our k-hop task), and—crucially—$w_1, \dots, w_l$ serve as additional arguments to the two-argument functions. For this more complicated task, Chen et al. (2024a) prove a depth-width tradeoff. Unlike our result, theirs does not make any assumption on the attention patterns, however, it is specifically proven for this more complicated task. Intuitively, separately presenting the functions $z_1, \dots, z_{l_1}$ from the entities $w_1, \dots, w_l$ serving as their second argument might play a key role in making the task challenging enough to enable the theoretical analysis in Chen et al. (2024a). Hence, it appears to remains open if such a bound can also be proven for a task directly matching k-hop reasoning (Figure 2).

Another line of work has shown limitations of *one-layer* transformers in performing function composition (Peng et al., 2024; Kozachinskiy et al., 2025); this is consistent with our evidence that k-hop tasks require increasing numbers of layers, but does not bound how many layers are needed.

Bounds from NC^1-hardness As transformers can be simulated in TC^0 (e.g. Merrill and Sabharwal, 2023; Strobl, 2023), some work has obtained bounds conditional on standard complexity conjecture $TC^0 \neq NC^1$. Assuming this conjecture, transformers generally cannot solve k-hop composition unless the number of layers increases as k increases; as an example, consider $\mathcal{E}$ to be $\{1, \dots, 5\}$, and $\mathcal{R}$ a generating subset of the alternating permutation group A_5; then solving k-hop composition is NC^1-hard and predicted to not be feasible for transformers. However, due to the difficulty of proving lower bounds for TC^0, this line of reasoning does not provide precise information about how quickly exactly L needs to grow with k.

Role of attention pattern Theorem 5.1 applies in the case where attention patterns do not depend on the input entity e (in fact, they might still depend on $r_1, \dots, r_k$). Our proof strategy makes use of this assumption, because it limits the amount of information that any individual attention head at the final positions can obtain about $r_1, \dots, r_k$. It remains open if this assumption can be relaxed.

[2] Indeed, it equals the Fibonacci number F_{k+2}. For large k, this is lower-bounded by $\left(\frac{3}{2}\right)^{k+2}$; to make it valid even for small k, it is sufficient to instead lower-bound by $\left(\frac{3}{2}\right)^k$.

[3] We note the connection to the general fact that every finite semigroup can be embedded into a finite semigroup generated by an idempotent and a nilpotent (note that f is nilpotent and g is idempotent), proven using a similar construction in Theorem 1.1 of Higgins (2017).

Intuitively, it does not seem clear how changing attention patterns could make the task easier. However, formally proving lower bounds for multi-layer transformer without either constraining attention patterns (as we do) or considering a more complex task (as done in Chen et al. (2024a)) remains challenging; we expect that further technical tools will be needed to overcome these challenges.

B Effect of more powerful models

Section 4 only presents results for our randomly initialized GPT-2 small model. Would the training data budget still grows in an exponentially way as the k increases, even with more powerful language models? We investigate this question by applying the same experiments in Section 4.2 with two other model setups: finetuning and scaling up model parameters. We only report 1-run results for all experiments in this section.

B.1 Finetuning

For finetuning setup, we finetune a pretrained language model on the same training set in Section 4 and evaluate it on the same test set. Here we start with the pretrained GPT-2 small model (Radford et al., 2019), and use the same hyperparameters as for our randomly initialized GPT-2. Note that the pretrained GPT-2 adopts a learned positional embedding (Vaswani et al., 2017) instead of RoPE (Su et al., 2024), and thus we cannot directly tell the effect of pretraining compared to non-pretrained model. Here we only use this experiment to confirm that the significant increase of data budget still holds for pretrained models.

Table 2 presents the results of pretrained GPT-2 small models. Overall, the data budget still exponentially grows as the k value increases. On k-hop$_{\textit{large}}$ the model needs $\times 10$ budget for 3-hoptask and $\times 100$ for 4-hop, which is the same as our randomly initialized transformer. The pretrained model achieves lower accuracy on k-hop$_{\textit{small}}$ datasets, e.g. only 93.7% accuracy on 2-hop task. Nonetheless, the model still learns to perfectly solve k-hop$_{\textit{small}}$datasets with enough data budget, e.g. for 3-hop, model accuracy gets significantly improved accuracy at $\times 5$ budget and reaches 100% at $\times 10$ budget. We consider the lower accuracy here is likely due to the lack of hyperparameter optimization and use of better position encoding.

B.2 Scaling up the model size.

We also evaluate setups where we scale up the number of model parameters. Kaplan et al. (2020) demonstrates that larger model size is crucial to gain high performance, especially the depth of transformer layers (Fagnou et al., 2024; Ye et al., 2024), and we want to investigate if larger models address the data budget issue. Here we experiment with same architecture described in Section 4.1 (i.e. GPT-2 with RoPE), and we set hyperparameters of architectures (e.g. number of layers, attention heads, etc.) according to the GPT-2 medium [4] and large model [5] configurations. We randomly initialize the model and train and evaluate it on the k-hop$_{\textit{small}}$ datasets in Section 4.1 from scratch. Hyperparameters for training are the same as Section 4.1.

Table 3 reports the results of such larger models. For both GPT-2 medium and large sized models, the growth of data budget issue still persists.

C Effect of model depth

The analysis in Section 5.3 theoretically proves that the depth of the model needs to grow linearly as the value k increases. In this section, we further provide empirical evidence to support our theory by showing that a transformer with fewer layers struggles with k-hop reasoning.

We conducted preliminary experiments in which we vary the model depth from 2 to 5 layers and use the k-hop$_{\textit{small}}$ datasets. We adopt the training data budgets that are sufficient for a 12-layer GPT-2-small model to succeed (see Table 1): $\times 1$ for 2-hop, $\times 5$ for 3-hop, and $\times 20$ for 4-hop. All other hyperparameters and training configurations are unchanged, except for the model depth (number of transformer layers).

We report test accuracy in Table 4. Noticeably, the required model depth grows as the hop number increases, which is consistent with what our theoretical analysis predicts.

D Dataset details

D.1 Datasets in Section 4

Namespaces of entity and relation names. We provide details on the entity and relation namespaces used to generate the datasets in Section 3.

[4]https://huggingface.co/openai-community/gpt2-medium
[5]https://huggingface.co/openai-community/gpt2-large

Dataset		Training data budget						
		×1	×2	×5	×10	×20	×50	×100
k-hop*small*	2-hop	93.7						
	3-hop	6.6	8.2	45.4	100			
	4-hop	5.1	6.5	6.9	8.4	23.8	100	100
k-hop*large*	2-hop	100						
	3-hop	2.8	2.5	3.9	87.5	100		
	4-hop	2.3	3	4.1	4.2	3.9	24.6	100

Table 2: Accuracy of finetuned GPT-2 small models on k-hop*small* and k-hop*large* datasets with different training data budgets.

Model	Dataset	Training data budget						
		×1	×2	×5	×10	×20	×50	×100
GPT-2 Medium	2-hop*small*	100						
	3-hop*small*	5.6	13.9	99.9	100			
	4-hop*small*	5	5.8	8	10.2	99.8	100	100
GPT-2 Large	2-hop*small*	100						
	3-hop*small*	6.5	22.0	100	100			
	4-hop*small*	4.6	5.6	7.9	11.0	99.6	100	100

Table 3: Accuracy of GPT-2 Medium and Large on k-hop*small* datasets with different training data budgets. Empty cells indicate that the data budget exceeds the number of available questions possible to generate.

Dataset	2 layers	3 layers	4 layers	5 layers
2-hop*small*	31	**96**	**97**	**100**
3-hop*small*	13	55	**100**	**100**
4-hop*small*	25	38	83	**95**

Table 4: Test accuracy with varying model depth. Accuracies above 90% are boldfaced.

Our dataset consists of $|\mathcal{E}|$ entities, each with a distinct name and N relations. We use 600 unique single-token person names (e.g., *Jennifer*) and 20 single-token relation names (e.g., *instructor*), generated by ChatGPT[6], as the namespaces for entities and relations, respectively. The complete vocabulary of relation and a subset of entity names are provided in Tables 6 and 7. Since our main experiments use a randomly initialized language model, the specific choice of vocabulary does not influence our conclusions.

Top-hierarchy entity profiles. Entities in the top layer of Figure 3 are not linked to any targets, making it non-trivial to generate their profiles. Nevertheless, we include their profiles in the training set to maintain consistency across all k-hop tasks with $k \in \{2, 3, 4\}$. In both the 2-hop and 3-hop tasks, answer tokens (i.e., entity names) appear in the training set as subject entities in their own profiles. To ensure the same holds for the 4-hop task, where answers correspond to top-layer entities, we generate profiles for these entities as well. Specifically, we generate these profiles by concatenating facts in which the subject entity is the top-layer entity itself, the relation is one from Table 6, and the object entity is a single-token name sampled from an additional set of 100 person names. These object names are distinct from the ones used in Figure 3. Since these facts are never used in any k-hop question in the training or test sets, including them does not affect our results or conclusions.

Table 5 reports the training set sizes for each dataset configuration. To maintain consistency across data budget setups, we include the same set of $|\mathcal{E}|$ entity profiles (e.g., $|\mathcal{E}|= 250$ profiles for k-hop*small*) in each training set. We partition the $|\mathcal{E}|$ entities into 5 disjoint subsets, each containing $|\mathcal{E}|/5$ entities, and only generate reasoning questions targeting one subset (e.g., entities in the bottom hierarchy of Figure 3). Each entity profile includes $|\mathcal{R}|$ relations (e.g., $|\mathcal{R}|= 10$ for k-hop*small*), allowing us to generate $|\mathcal{R}|^2 \times\, |\mathcal{E}|/5 = 5000$ questions for 2-hop*small*, of which 80% are randomly selected as training instances.

D.2 Datasets for mixed and curriculum learning

In mixed learning, we introduce lower-hop reasoning questions as auxiliary training instances to facilitate learning more complex reasoning tasks. For the 3-hop task, we add 2-hop instances, and

[6] https://chatgpt.com/

Dataset	Hop	×1	×2	×5	×10	×20	×50	×100
small	2-hop	4 250						
	3-hop	4 250	8 250	20 250	40 250			
	4-hop	4 250	8 250	20 250	40 250	80 250	200 250	400 250
large	2-hop	20 500						
	3-hop	20 500	40 500	100 500	200 500	400 500		
	4-hop	20 500	40 500	100 500	200 500	400 500	1 000 500	2 000 500

Table 5: Statistics of the number of training instances in each setup.

instructor	teacher	ruler	advisor
supervisor	leader	manager	director
patron	mentor	administrator	coordinator
tutor	predecessor	sponsor	financier
backer	overseer	employer	boss

Table 6: Vocabulary of relation names.

Emil	Gavin	Chad	Flora
Adam	Addie	Bobby	Edwin
Gabby	Helen	Jeffery	Joel
Kris	Kristen	Lisa	Liam
Eva	Emma	Dylan	Isabella

Table 7: Subset of vocabulary of entity names.

for the 4-hop task, we add both 2-hop and 3-hop instances. For k-hop$_{small}$, we include 4k 2-hop instances as auxiliary data for 3-hop$_{small}$, and 4k 2-hop and 20k 3-hop instances for 4-hop$_{small}$. For k-hop$_{large}$, we include 32k 2-hop instances for 3-hop$_{large}$, and 32k 2-hop and 100k 3-hop instances for 4-hop$_{large}$. Due to computational constraints, we did not specifically tune the size of auxiliary data. The curriculum learning setup uses the same auxiliary instances as mixed learning.

E Training details

E.1 Baseline

This section provides the model architecture and training setup used in Section 4. Unless stated otherwise, the same configuration is applied across all experiments in this paper.

Model architecture. We adopt the GPT-2 small architecture[7], consisting of 12 transformer layers with 12 attention heads. The input embedding dimension is 768, and the context window is limited to 1024 tokens. Instead of absolute position embeddings used in the original Transformer (Vaswani et al., 2017), we employ Rotary Position Embedding (RoPE) (Su et al., 2024) to encode positional information. We use the default GPT-2 tokenizer and extend the vocabulary to include all entity profile names (e.g., *Jennifer*), resulting in a vocabulary size of $|V| = 50,740$.

[7]https://huggingface.co/openai-community/gpt2

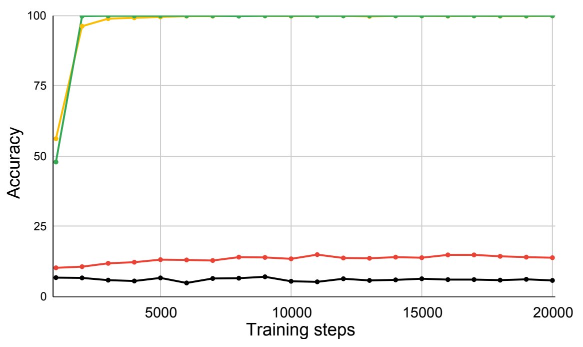
Training. The batch size is set to 512 with gradient accumulation steps of 4. We use the AdamW optimizer (Kingma and Ba, 2015; Loshchilov and Hutter, 2019) with the following hyperparameters: learning rate of $5e-4$, $\epsilon = 1e-6$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight decay of 0.1. Training begins with a 1k-step warm-up phase, followed by a cosine learning rate scheduler (Loshchilov and Hutter, 2016), with a minimum learning rate set to $0.1\times$ the initial learning rate.

Experiments are run on Nvidia A100 and H100 GPU cards (80GB). Each experiment is conducted on one single GPU, which takes about 8 hours for 20k optimization steps. The implementation is based on Huggingface (Wolf et al., 2019) and Pytorch (Paszke, 2019). GPT-2 is released under the MIT License by OpenAI.

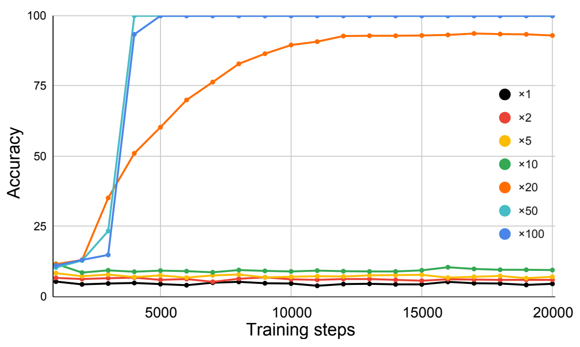
E.2 Setup for mixed and curriculum learning

The model architecture for mixed and curriculum learning experiments remains the same as the baseline configuration described in Section E.1. The training setup for mixed learning also follows the baseline training setup without any modifications.

Training in curriculum learning is divided into multiple stages, where each stage progressively introduces harder reasoning tasks. For a k-hop task, training consists of $k-1$ stages: The first stage includes only 2-hop questions. The second stage adds 3-hop questions. The third stage adds 4-hop questions to the training set (only applicable for 4-hop tasks). Hence we have 2 training stages for 3-hop task and 3 training stages for 4-hop task. The maximum number of training steps for each stage across different target tasks is reported in Table 8. Each stage employs the same learning rate scheduler and warm-up steps as in the baseline training setup to maintain consistency. The batch size and gradient accumulation steps remain the same as in the baseline setup.

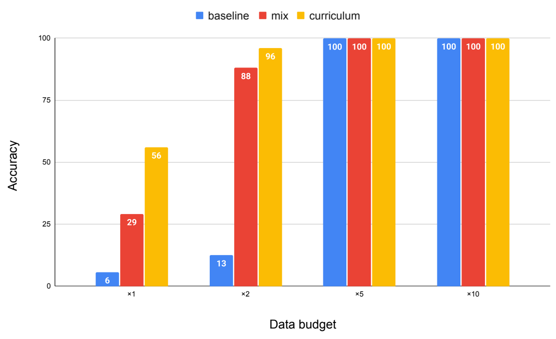


(a) 3-hop$_{small}$

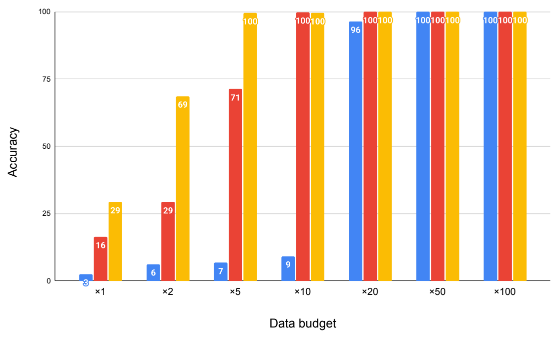


(b) 4-hop$_{small}$

Figure 9: Model accuracy on 3-hop$_{small}$ and 4-hop$_{small}$. x-axis refers to the number optimization steps.



(a) 3-hop$_{small}$



(b) 4-hop$_{small}$

Figure 10: Model performance on k-hop$_{small}$ datasets with mixed learning and curriculum learning.

Task	Stage 1	Stage 2	Stage 3	Total
3-hop$_{small}$	10000	10000	-	20000
4-hop$_{small}$	5000	5000	10000	20000
3-hop$_{large}$	10000	10000	-	20000
4-hop$_{large}$	10000	10000	20000	40000

Table 8: Training steps for each training stage of curriculum learning

F Detailed results

F.1 Results for LMs on k-hop$_{small}$

We plot the test accuracy of LMs on k-hop$_{small}$ across training steps in Figure 9. The pattern is similar to the one observed in Figure 4. Models trained with small budgets only give modest improvement over random baseline (i.e. 2% for k-hop$_{small}$). Larger budgets not only lead to higher accuracy, but also achieves this with much less training steps.

We also report k-hop$_{small}$ results of models trained with mixed learning and curriculum learning in Figure 10. Still, we observe that curriculum learning gives the best result compared to the baseline and mixed learning.

F.2 Standard deviation

For each experiment reported in Section 4 and 6, we made 3 runs based on different random seeds. We report the mean and standard deviation of the test accuracy for each model in Table 9. For most results we do not observe a large standard deviation, indicating that our conclusion is robust to the randomness. For particular runs there is a large deviation, especially when the data budget is not enough (e.g. model trained with curriculum learning on 4-hop$_{large}$with $\times 2$ budget), which gets smaller when we further add more data into the training set.

F.3 Log scale of data budget

We plot the minimal data budget required to solve k-hop tasks on a log scale as k increases. The data points are based on numbers in Table 1. Figure 11 shows the results, confirming that the required data budget grows exponentially with k.

G Additional mechanistic interpretability experiments

G.1 Patching preceding prompt tokens

Figure 6 suggests that only the last token (e.g. the whitespace <*space*>) includes information about

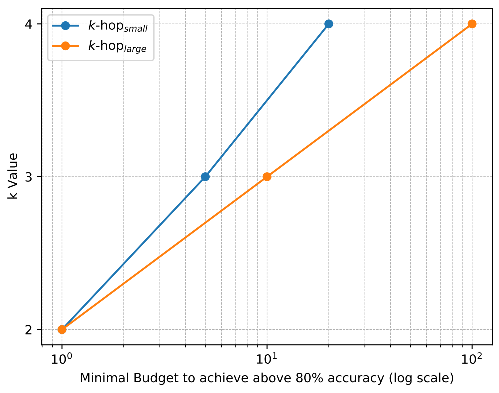


Figure 11: Minimal data budget to solve k-hop tasks.

all bridge entities, and hence the reasoning process likely occurs at this position. In this section, we use activation patching to further demonstrate that the reasoning process of our language model only occurs at the last token position instead of preceding prompt tokens.

Our activation patching still addresses three types of runs: clean run, corrupted run and patched run. For each clean run, we randomly select a distinct instance as the corrupted run. For each layer and each token position in the input prompt, we create a patched run by replacing the residual stream of the clean run with that of the corrupted run at the corresponding position. The causal effect is calculated as $P_{\text{clean}} - P_{\text{patched}}$, where P_{clean} denotes the output probability of the correct answer in the clean run, and P_{patched} denotes the probability in the patched run. We report the average causal effect over 1000 held-out instances.

Figure 12 presents the activation patching results across token positions. Noticeably, no significant causal effects are observed in any token positions following the <*Entity*> token, except for the last <*space*> token. Since the <*Entity*> token is the first position where the model can access complete query information (i.e., relations and source entity), this result supports our claim that the reasoning process primarily occurs at the last token position.

We also observe large causal effects on relation tokens when patching deeper layers (e.g., the 4th layer for the <*r4*> token). We consider this effect is because the model only start to read the information of <*r4*> relation since the 4th layer when predicting the answer. Hence, deeper layers of <*r4*> position should not involve any reasoning-related computation. To show this, we also perform the same activation patching experiment by replacing only the output of each MLP layer. As shown in Figure 13, the relation tokens only show causal effects in the first layer, further supporting our hypothesis that deeper layers do not reprocess relation information.

G.2 Causal effects across training steps

In Section 5, we observed that LMs learn to solve k-hop tasks through a layer-wise lookup process, with specific layers responsible for producing bridge entities from 1-hop to k-hop. A key question is whether these circuits (i.e., layers) are developed sequentially from 1-hop to k-hop or simultaneously across multiple hop positions during training. To investigate this, we apply the activation patching experiment described in Section 5 at every checkpoint of the training process.

We focus on the model trained on 4-hop$_{large}$ with a $\times 100$ budget, following the setup in Section 5. Checkpoints are saved every 1k training steps, and we apply activation patching at the last input token position. For each checkpoint, we measure the causal effect of each layer for bridge entities at each hop position.

LMs tend to build circuits of different i-hop bridge entities simultaneously. Figure 14 shows the causal effect of each layer across training steps. We observe that circuits responsible for 1-hop, 2-hop, and 3-hop bridge entities emerge simultaneously at around the 17000th training step, with each circuit appearing in distinct layers (e.g., the 1st layer for 1-hop entity). This pattern indicates that the model tends to develop circuits for different hop positions at once rather than sequentially from easier (e.g., 1-hop) to more complex (e.g., 3-hop) entities.

Curriculum learning gradually build circuits on existing ones. We further analyze the development of circuits in the curriculum learning model trained on the 4-hop task with a $\times 5$ budget (Section 6). Training this model includes 3 stages. Checkpoints are saved every 1k steps, and causal effects are calculated at each stage. For each stage, we calculate the causal effects using the following corrupted runs:

- $C_{\text{1-hop}}$: Assesses 1-hop circuits across stage 1, 2 and 3.
- $C_{\text{2-hop}}$: Assesses 2-hop circuits across stages 2 and 3.
- $C_{\text{3-hop}}$: Assesses 3-hop circuits in stage 3.

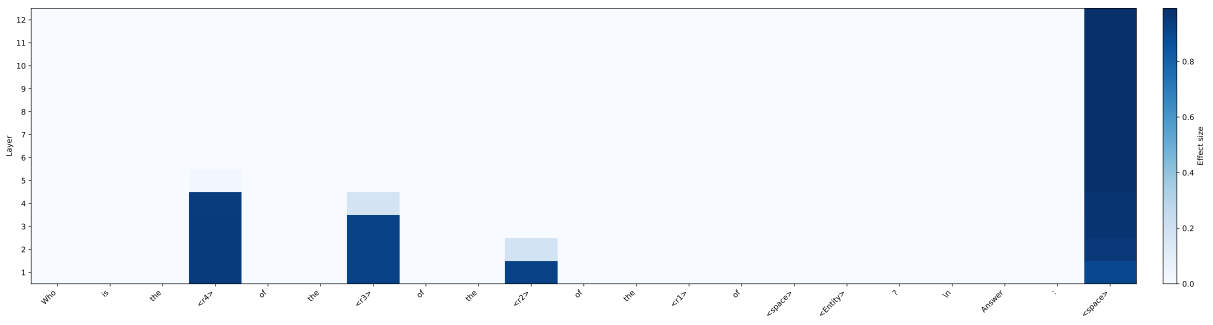


Figure 12: Results for activation patching replacing the residual stream of a particular layer across prompt tokens.

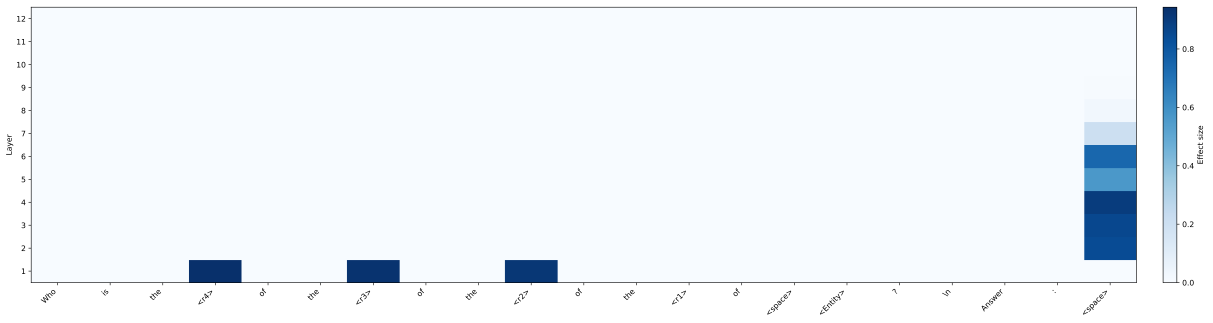


Figure 13: Results for activation patching replacing the MLP output of a particular layer across prompt tokens.

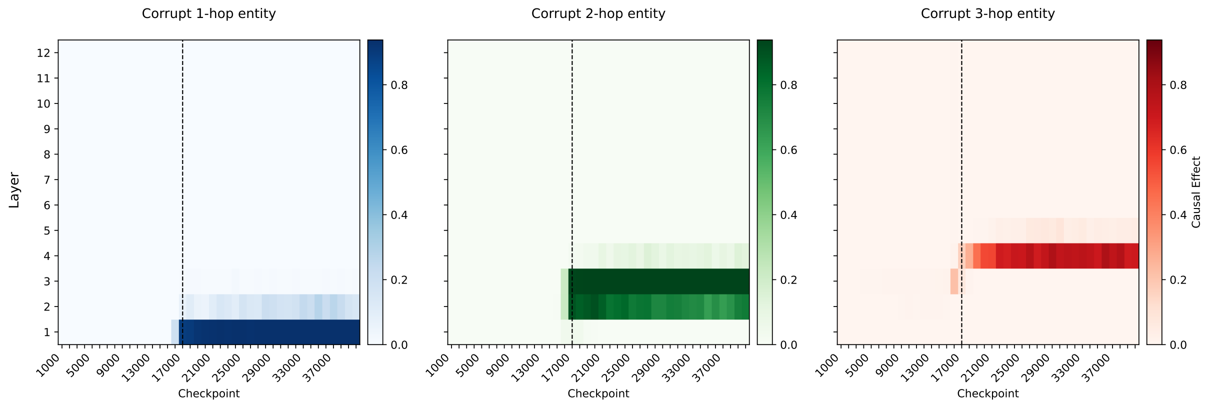


Figure 14: Causal effects calculated by corrupting 1-hop, 2-hop and 3-hop bridge entity in our *baseline* model. x-axis refers to checkpoints across training steps. We observe that the circuits corresponding to different hop positions tend to emerge at once (e.g., around the 17,000th step), rather than gradually developing over time.

Figure 15 presents the results with our curriculum learning model. During stage 1, the model establishes circuits for 1-hop entities. In stage 2, the 2-hop circuit emerges, building upon the existing 1-hop circuit. Stage 3 follows the same pattern, with the 3-hop circuit extending the prior circuits. This layer-by-layer construction supports our hypothesis that curriculum learning encourages progressive circuit development, allowing higher-hop circuits to build upon existing lower-hop circuits, explaining the observed effectiveness in Section 6.

Curriculum learning has also been explored in prior work (Deng et al., 2024; Hao et al., 2024), where the focus is on internalizing explicit reasoning abilities. These studies start from chain-of-thought (CoT) rationales and train language models to reason with progressively fewer prompt tokens. In contrast, our setup does not rely on any explicit rationales. Instead, we study how curriculum learning affects the data budget required for training and provide an explanation for why such strategies improve sample efficiency from a mechanistic interpretability perspective.

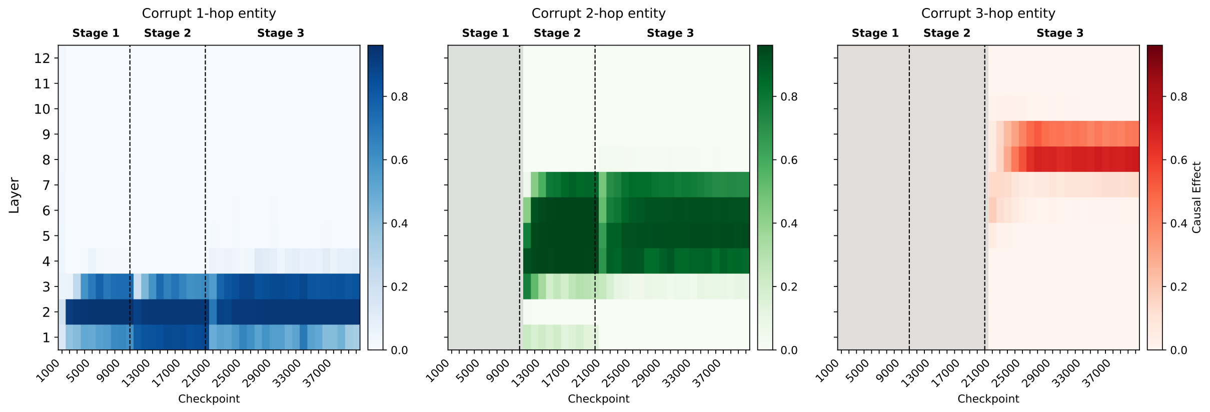


Figure 15: Causal effects calculated by corrupting 1-hop, 2-hop and 3-hop bridge entity in our *curriculum learning* model. x-axis refers to checkpoints across training steps. Gray regions indicate stages where causal effects are not calculated for certain entities, e.g., stage 1 does not include 3-hop bridge entities in the training data, so the rightmost figure omits these effects in stage 1. Circuits for higher-hop entities tend to be established on top of existing ones for lower-hop entities.

Model	Size	Task	×1		×2		×5		×10		×20		×50		×100	
			Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
baseline	small	2-hop	99.8	0.1												
		3-hop	5.7	0.0	12.6	2.6	99.9	0.1	100.0	0.0						
		4-hop	4.3	0.4	6.2	0.5	6.7	0.2	9.2	0.4	96.4	3.1	96.4	0.0	100.0	0.0
	large	2-hop	99.9	0.0												
		3-hop	2.5	0.3	3.1	0.1	4.9	1.6	94.6	9.4	100.0	0.0				
		4-hop	2.0	0.3	2.6	0.3	3.1	0.1	3.7	0.3	4.0	0.4	6.3	1.0	100.0	0.0
mix	small	2-hop	100.0	0.0												
		3-hop	29.2	3.0	88.1	8.6	99.9	0.1	100.0	0.0						
		4-hop	16.4	1.8	29.3	1.8	71.3	41.4	99.8	0.1	100.0	0.0	100.0	0.0	100.0	0.0
	large	2-hop	100.0	0.0												
		3-hop	8.3	1.4	11.2	3.8	38.7	18.1	100.0	0.0	100.0	0.0				
		4-hop	2.1	0.2	2.7	0.1	3.7	0.2	3.4	0.1	4.3	0.6	7.2	1.9	100.0	0.0
curriculum	small	2-hop	100.0	0.0												
		3-hop	56.1	1.5	96.0	0.7	100.0	0.0	100.0	0.0						
		4-hop	29.3	2.7	68.7	5.4	99.6	0.2	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
	large	2-hop	100.0	0.0												
		3-hop	35.3	1.5	96.3	1.2	100.0	0.0	100.0	0.0	100.0	0.0				
		4-hop	9.4	1.9	36.1	14.8	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0

Table 9: Accuracy (mean ± std) for 2-/3-/4-hop tasks under varying data budgets. Blank cells denote that the data budget exceeds the number of available questions.