

Library-Like Behavior In Language Models is Enhanced by Self-Referencing Causal Cycles

Munachiso Nwadike^{1,2}, Zangir Iklassov¹, Toluwani Aremu¹,
 Tatsuya Hiraoka^{1,2}, Benjamin Heinzerling^{2,3}, Velibor Bojkovic¹,
 Hilal Alqaubeh^{1,2}, Martin Takáč¹, Kentaro Inui^{1,2,3}

¹MBZUAI, Abu Dhabi, UAE • ²RIKEN AIP, Japan • ³Tohoku University, Japan

munachiso.nwadike@mbzuai.ac.ae

Abstract

We introduce the concept of the *self-referencing causal cycle* (abbreviated RECALL)—a mechanism that enables large language models (LLMs) to bypass the limitations of unidirectional causality, which underlies a phenomenon known as the *reversal curse*. When an LLM is prompted with sequential data, it often fails to recall preceding context. For example, when we ask an LLM to recall the line preceding “O say does that star-spangled banner yet wave” in the U.S. National Anthem, it often fails to correctly return “Gave proof through the night that our flag was still there”—this is due to the reversal curse. It occurs because language models such as ChatGPT and Llama generate text based on preceding tokens, requiring facts to be learned and reproduced in a consistent token order. While the reversal curse is often viewed as a limitation, we offer evidence of an alternative view: it is not always an obstacle in practice. We find that RECALL is driven by what we designate as *cycle tokens*—sequences that connect different parts of the training data, enabling recall of preceding tokens from succeeding ones. Through rigorous probabilistic formalization and controlled experiments, we demonstrate how the cycles they induce influence a model’s ability to reproduce information. To facilitate reproducibility, we provide our code and experimental details at <https://github.com/samunaai/remember>.

1 Introduction

Consider, by way of metaphor, a large language model (LLM) as the parametric equivalent of a physical library of knowledge (Lederman and Mahowald, 2024). A library evokes structured collections of books or documents, each cataloged for efficient retrieval. Similarly, pretraining LLMs on billions of tokens transforms them into repositories of encoded knowledge (Petroni et al., 2019;

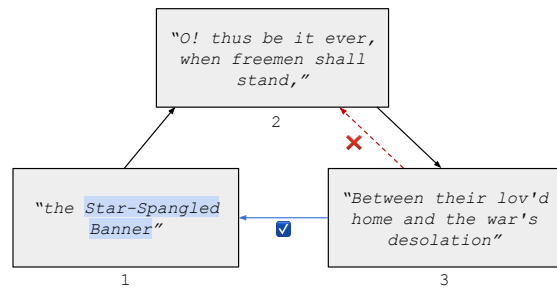


Figure 1: In a text containing the U.S. National Anthem, the phrase “Star-Spangled Banner” appears several times and functions as a cycle token-sequence, creating a loop termed a *self-referencing causal cycle*. This cycle reconnects later parts of the text to earlier ones. Without these cycle tokens, the model would be unable to predict the preceding line “O! thus be it ever, when freemen shall stand” from the succeeding line “Between their lov’d home and the war’s desolation” due to left-to-right causality (the red line). The blue line indicates that the cycle token-sequence occurs at multiple points in the text, acting as a reference point for the model to retrieve preceding context.

Heinzerling and Inui, 2020; Wang et al., 2024). Prompts, therefore, act as cross-references, directing retrieval of specific information, much like library indexes facilitate access to books on shelves.

In a library, we expect to retrieve information reliably. However, language models are not always suited for this. For example, when asked to recall the line preceding “Between their lov’d home and the war’s desolation” in the U.S. National Anthem, a language model often fails to return “O! thus be it ever, when freemen shall stand” (see Figure 1). This is a symptom of the “reversal curse,” where models struggle to generalize reversed relationships from their training data.

In this work, we explore a mechanism within LLMs that naturally mitigates the reversal curse leveraging inherently occurring patterns in pre-training data. Specifically, we show how self-

referencing causal cycles (RECALL) emerge from repeated token sequences, allowing models to bypass this limitation without requiring explicit in-context reversal strategies. These cycles, induced by what we term *cycle tokens*, act as natural hyperlinks between different parts of a text, enhancing memory retrieval. Our analysis focuses on autoregressive models, for which we propose a novel two-step RECALL process to improve information retrieval.

2 Related Work

The *reversal curse*, or “inverse search problem,” has been extensively studied in large language models. Early works by Berglund et al. (2023) and Allen-Zhu and Li (2023) identify this issue, showing that models struggle to generalize even simple inverse statements, most notably the identity form “A is B \rightarrow B is A.” We adopt a broader framing, exemplified by “A after B \rightarrow B before A,” which relies purely on token order and subsumes the identity case as a special instance where A and B are identical. This limitation, rooted in the autoregressive nature of models like GPT (Achiam et al., 2023) and LLaMA (Dubey et al., 2024), persists even in those equipped with chain-of-thought reasoning (Guo et al., 2024).

Recent studies highlight how reverse thinking enhances reasoning abilities (Chen et al., 2024), leveraging in-context learning to address tasks such as arithmetic and logical deduction. However, it is already known that the reversal curse does not manifest in in-context learning settings (Berglund et al., 2023), as models benefit from explicit contextual information during inference.

Bidirectional models, like BERT (Devlin et al., 2019), avoid the reversal curse by using masked token objectives (Wu et al., 2024), allowing them to reason about context in both directions. However, these models are not designed for autoregressive tasks such as next-token prediction, which underpins state-of-the-art chatbots.

Methods to mitigate the reversal curse often involve data augmentation. For example, Guo et al. (2024) and Golovneva et al. (2024) explore token permutation techniques, while Springer et al. (2024) propose token repetition to enhance causal links in training data. Unlike these manual interventions, our work investigates naturally occurring patterns in pretraining data, and how they organically mitigate the reversal curse.

3 Formalizing RECALL

This section formalizes RECALL by introducing the concept of “cycle tokens” and their causal effects. To establish this foundation, we begin by revisiting the reversal curse in probabilistic terms.

3.1 Revisiting the Reversal Curse

Consider the set of all possible token sequences, denoted by \mathcal{S} , for a given textual dataset that adheres to a true data distribution P . For simplicity, we assume the language is written in a left-to-right (LTR) script, such as English. However, the arguments generalize naturally to other text directions, such as right-to-left (RTL, e.g., Arabic) or top-to-bottom (TTB, e.g., Japanese), requiring only minor notational modifications.

Let $\mathcal{S}_{\text{seq}} \in \mathcal{S}$ be a sequence of n tokens, represented as $\mathcal{S}_{\text{seq}} := [e_1, e_2, \dots, e_n]$. We partition \mathcal{S}_{seq} into two segments at some index i , where $S_l := [e_1, e_2, \dots, e_i]$ represents the left-hand segment, and $S_r := [e_{i+1}, \dots, e_n]$ represents the right-hand segment. Probabilistically, S_r is the most likely continuation of S_l under the distribution P . For example, S_r could correspond to the continuation “*O say does that star-spangled banner yet wave,*” while S_l represents the preceding context “*Gave proof through the night that our flag was still there.*” Identifying that the latter precedes the former poses a challenge for autoregressive models. The reason for this can be illustrated as follows:

For an autoregressive model \mathcal{M} trained on the true data distribution, we have $P \approx P_{\mathcal{M}}$, where $P_{\mathcal{M}}$ denotes the model’s learned approximation of the true data distribution. Accordingly, we expect:

$$S_r = \arg \max_{s \in \mathcal{S}} P_{\mathcal{M}}(s|S_l). \quad (1)$$

This implies that \mathcal{M} can readily produce the highest-probability right-hand sequence S_r given a left-hand sequence S_l . However, the model struggles to select the correct S_l given S_r , because:

$$S_l \neq \arg \max_{s \in \mathcal{S}} P_{\mathcal{M}}(s|S_r). \quad (2)$$

Instead, the model can only indirectly compute the left-hand sequence using Bayes’ rule:

$$S_l = \arg \max_{s \in \mathcal{S}} P_{\mathcal{M}}(S_r|s)P_{\mathcal{M}}(s). \quad (3)$$

Equation (3) is explained in greater detail in Appendix B.1.

However, computing the argmax in equation 3 would require iterating through all possible sequences $s \in \mathcal{S}$, pairing each s with the fixed S_r , and then evaluating the combined sequence $[s, S_r]$ using the model \mathcal{M} to obtain $P_{\mathcal{M}}(S_r|s)$ and $P_{\mathcal{M}}(s)$.

While the model can easily compute $P_{\mathcal{M}}(S_r|s)$, which represents the likelihood of the right sequence given the left part, and $P_{\mathcal{M}}(s)$, the prior probability of a token sequence, iterating over all possible sequences $s \in \mathcal{S}$ is computationally infeasible due to the combinatorial growth of possible sequences.

To address this, we would like to narrow the search to a smaller candidate set S_{l_c} , generated by prompting the right-hand sequence:

$$S_l = \arg \max_{s \in S_{l_c}} P_{\mathcal{M}}(S_r|s) P_{\mathcal{M}}(s). \quad (4)$$

However, how do we construct S_{l_c} if S_r offers no direct clues about its preceding sequence? We would require knowledge of the sequence to the left S_l (left-of-left), which we do not have.

3.2 Introducing Cycle Tokens

Our core hypothesis is that, instead of predicting S_l directly from S_r , we construct a modified sequence $S'_r := [e_{i+1}, \dots, e_n, e_1]$ by appending e_1 to the end of S_r . This modified sequence serves as a pointer back to the start of the original sequence, providing access to S_l . The token e_1 acts as a *cycle token*—so named because it induces a cycle in the causal flow of next-token prediction, enabling the model to effectively “see” left-hand tokens from the right-hand side.

From this modified sequence, we can extract $S'_l := [e_2, e_3, \dots, e_i]$ using continued next-token predictions. Importantly, the entire sequence S_r does not need to be repeated; even a single cycle token can serve as a pointer, creating what we term a *self-referencing causal cycle* (see Figure 2).

As the length i of S_l grows, S'_l increasingly resembles S_l . For example, consider two sentences of 100 words each and two sentences of 5 words each. With one differing letter in each pair, the longer sentences exhibit greater similarity. Formally:

$$S'_r := S_r \oplus e_1 \quad \text{and} \quad S_l := e_1 \oplus S'_l,$$

where \oplus denotes concatenation. Therefore:

$$S_l \approx S'_l. \quad (5)$$

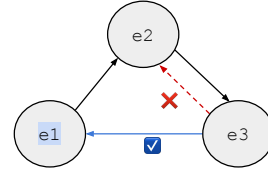


Figure 2: A token sequence $[e_1, e_2, e_3]$ illustrates the difficulty of predicting e_2 from e_3 due to left-to-right causality. By appending token e_1 to the end of the sequence, a loop is formed, allowing the model to transition from e_3 back to e_2 . Red and blue indicate the same concepts as in Figure 1.

Noting that:

$$S'_l \rightarrow S_l \quad \text{as} \quad i \rightarrow \infty.$$

Thus, in the presence of a self-referencing causal cycle, a left-to-right autoregressive model can approximate left-hand sequence information from the right-hand side. Interpreted intuitively, the self-referencing causal cycle is a mechanism that allows the model to ‘loop back’ and access earlier parts of sequences of any length, thereby introducing a bidirectional influence to a unidirectional model.

4 Experiments and Analysis

4.1 Deterministic Few-Token RECALL

To demonstrate self-referencing causal cycles, we use simple few-token datasets and a small decoder-based transformer model (Vaswani, 2017) with two layers and eight attention heads. The datasets are designed for ease of interpretation. Implementation details are provided in Appendix D.

Suppose we train our model on four-token sequences of the form $[e_1, e_2, e_3, e_1]$. Let e_1 be an integer randomly selected from $[1, 100]$, e_2 from $[101, 200]$, and e_3 from $[201, 300]$. If each integer is randomly selected, such that each e_1 can only be paired with a unique e_2 , and each e_2 with a unique e_3 , then the dataset will consist of 100 samples. For example, one possible training sample is $S_{train} = [79, 155, 264, 79]$, where e_1 appears twice—at the beginning and the end of the sequence. During training, the transformer memorizes this sequence, and we test whether it can predict the sequence $S_{test} = [264, 79, 155]$. If the model can predict this sequence S_{test} with 100% accuracy, it demonstrates that it can recover 155 from 264 by using the token 79 as a “cycle” to link the end of the sequence back to its beginning. In this case, token 79 serves as the cycle token.

Experiment	Memorized Sequence	Reversal Path	Viable
Baseline	(e1, e2, e3, e1)	e3 → e1 → e2	✓
	(e1, E2, E3, e1)	E3 → e1 → E2	✓
Length of Path	(e1, e2, e3, E4, e1)	e3 → E4 → e1 → e2	✓
	(e1, E2, E3, E4, e1)	E3 → E4 → e1 → E2	✓
Length of 'Out-of' Path	(e1, e2, E3, e4, e1)	e4 → e1 → e2	✓
	(e1, E2, E3, E4, e1)	E4 → e1 → E2	✓
Cycle Composability	(e1, e2, e3) and (e3, e1, e4)	e3 → e1 → e2	✗
	(e1, E2, E3) and (E3, e1, E4)	E3 → e1 → E2	✗
Hyperlink Composability	(e5, e3, e1, e4) and (e0, e1, e2, e3)	e2 → e3 → e1 → e4	✓
	(E5, E3, e1, E4) and (E0, e1, E2, E3)	E2 → E3 → e1 → E4	✓

Table 1: List of the few-token experiments showing the sequences, causal paths, and the viability of reversal. These experiments explore various configurations, including baseline setups, variations in path length, out-of-path noise, and cross-sample hyperlinks, to evaluate the effectiveness of cycle tokens in linking sequences. $e1$ is the cycle token in all cases, with the exception of Hyperlink Composability, where $e3$ and $E3$ are the respective cycle token and token-sequences.

In all experiments, we consider transitions from a right-hand *token* (denoted lowercase e) to a left-hand *token*, and from one *sequence* (denoted uppercase E) to another. For example, instead of memorizing $\mathcal{S}_{\text{train}} = [e1, e2, e3, e1]$, we could memorize $\mathcal{S}_{\text{train}} = [e1, E2, E3, e1]$ and test recovery of $\mathcal{S}_{\text{test}} = [E3, e1, E2]$. Such sequence-to-sequence experiments better reflect real-world pretraining data, where information spans phrases or sentences.

The settings described above form our **Baseline** few-token experiment, outlined in Table 1. We introduce several variations on this experiment to test the robustness of self-referencing causal cycles. For instance, instead of transitioning directly from $e3$ to $e1$, we can introduce a sequence of “noise” tokens $E4$ in between. If $E4$ has a length $\mathcal{N} = 3$, there are three tokens blocking the transition path from the start token $e3$ to the cycle token $e1$. We term this the **Length of Path** experiment, as we vary the length \mathcal{N} (see Figure 4). Alternatively, we could include a spurious sequence $E3$ between two tokens, $e4$ and $e2$, in the right-to-left direction but not in the left-to-right direction (i.e., not in the transition path from $e4$ to the cycle token). We term this the **Length of ‘Out-of’ Path** experiment. If the model memorizes all such out-of-path sequences with perfect validation accuracy, it reinforces the

evidence that $e1$ is responsible for creating the self-referencing causal cycle.

Another characteristic of real-world data is that token-sequences carrying related information may be distributed across samples in separate data batches. The **Hyperlink Composability** experiment demonstrates that a cycle token can hyperlink different samples, enabling causal “jumps” between them. For instance, if we have two samples in different batches, $\mathcal{S}_1 = [e5, e3, e1, e2]$ and $\mathcal{S}_2 = [e0, e1, e2, e3]$, we can “jump backwards” and recover $\mathcal{S}_{\text{test}} = [e2, e3, e1, e2]$ where $e3$ acts as a bridge token.

RECALL is achievable • In nearly all cases, we observe perfect generalization after training for a specified number of steps, as shown in Figure 3.

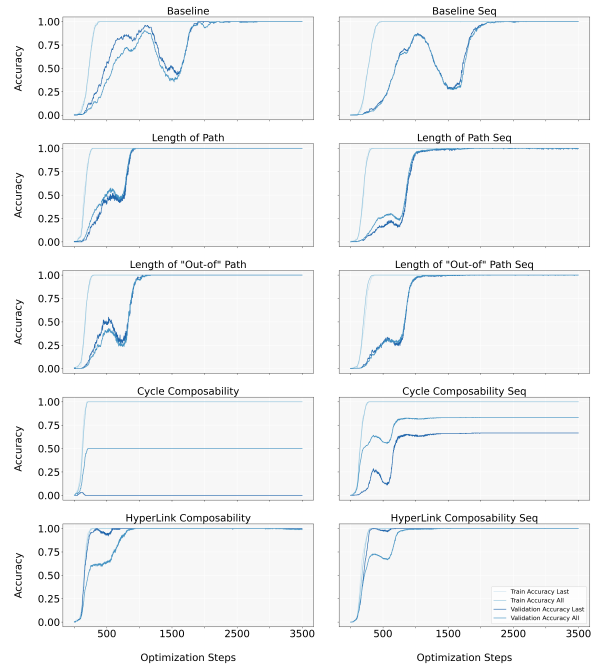


Figure 3: Performance of cycle tokens in few-token experiments, showing validation accuracy as a function of optimization steps (epochs). Train-validation splits follow Figure 1. Cycle tokens enable transitions from the starting to the target token-sequence, bypassing left-to-right causality. Last accuracy refers to correct prediction of the final token-sequence—the base case of which is a single token—while All accuracy refers to correct prediction of all subsequent tokens after the first. For example, in a validation sequence $(e3, e1, e2)$, All means correctly predicting $e1$ and $e2$, while Last refers to predicting $e2$. In a mixed-sequence case like $(E3, e1, E2)$, All requires correctly predicting both $e1$ and $E2$, and Last refers to the correct prediction of $E2$. In all scenarios, 100% accuracy is achieved in predicting the left-hand sequence using the cycle token as a bridge.

The sole exception is the Cycle Composability experiment, which reveals that cycle tokens function more effectively when the left-hand context does not alter their semantics. For example, consider a training dataset consisting of $\mathcal{S}_1 = [e1, e2, e3]$ and $\mathcal{S}_2 = [e3, e1, e4]$, and we seek to transition from $e3$ to $e2$ (i.e., $\mathcal{S}_{test} = [e3, e1, e2]$). If $e3$ were not present in \mathcal{S}_2 , a transition from $e1$ to $e2$ would be possible by virtue of Hyperlink Composability. In this case, the model would have a 50% chance of predicting $e2$ following $e1$ (a stochastic setting discussed further in Section 4.2). However, because \mathcal{S}_2 is present in the training data, the model will be inclined to predict $e4$ after seeing $[e3, e1]$ 100% of the time, as $[e3, e1, e4]$ represents a pattern trained into it via cross entropy loss. This behavior is desirable, as it demonstrates how self-attention naturally alters the semantic interpretation of a token based on preceding context.

RECALL is consistent over varying sequence lengths \mathcal{N} • In particular, we demonstrate that the experiments in 3, which use sequence length $\mathcal{N} = 3$, do not imply any loss of generality. We demonstrate this by virtue of ablation studies with varying sequence lengths, shown in Figure 4.

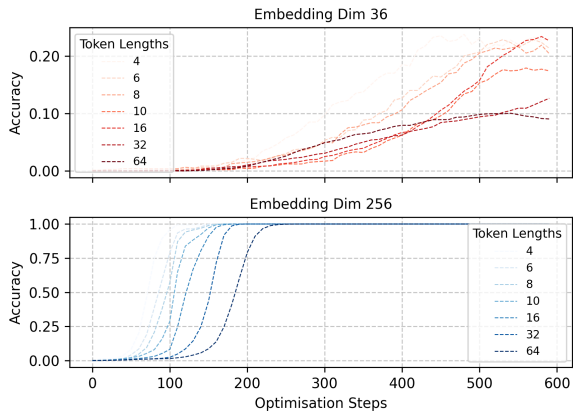


Figure 4: Increasing the value of \mathcal{N} results in longer token-sequences, but validation accuracy remains high with sufficient embedding dimensionality.

Specifically, we examine the effect of increasing the path length between the starting token and the cycle token from $\mathcal{N} = 4$ to $\mathcal{N} = 64$. When keeping the default embedding dimension of 36 (as in Figure 3), generalization slows as sequence length increases. However, increasing the embedding dimension to 256 enables generalization across all token lengths within significantly fewer optimization steps. These findings align with prior work on the boundary between grokking and memorization

in language models (Liu et al., 2022), suggesting that varying sequence lengths would only introduce an added step of hyperparameter tuning.

In summary, the deterministic few-token experiments of Section 4.1 provide initial evidence that self-referencing causal cycles, created by cycle tokens, enable models to effectively recall left-hand sequences and mitigate the reversal curse, as was outlined theoretically in Section 3.2.

4.2 Stochastic Few-Token RECALL

The experiments in Section 4.1 assumed that each cycle token-sequence was followed by a unique token-sequence. For instance, in the single-token Hyperlink Composability experiment (Table 1), each $e1$ was succeeded by a unique $e4$ token.

Experiment	Memorized Sequence	Reversal Path	Viable
Direct Stochasticity	$(e1, \{e2_i\}_{i=1}^n, e3, e1)$	$e3 \rightarrow e1 \rightarrow e2_i$	✓
	$(e1, \{E2_i\}_{i=1}^n, E3, e1)$	$E3 \rightarrow e1 \rightarrow E2_i$	✓
Hyperlink Stochasticity	$(\{e5_i\}_{i=1}^n, e3, e1, \{e4_i\}_{i=1}^n)$ and $(e0, e1, e2, e3)$	$e2 \rightarrow e3 \rightarrow e1 \rightarrow e4_i$	✓
	$(\{E5_i\}_{i=1}^n, E3, e1, \{E4_i\}_{i=1}^n)$ and $(E0, e1, E2, E3)$	$E2 \rightarrow E3 \rightarrow e1 \rightarrow E4_i$	✓

Table 2: Experiment capturing self-referencing causal cycle path under stochastic setting. The figure illustrates how stochastic conditions affect the model’s recall of a target token from a candidate set. During training, a range of valid candidates is presented to the model. In the case of direct stochasticity, the candidate set is denoted as $e2_{i=1}^n$, where multiple valid values for $e2$ can be sampled. During testing, the objective is to determine whether the model can correctly retrieve a specific target token $e2_i$ from the indexed set $[1, n]$. This experiment highlights the model’s handling of ambiguity introduced by stochastic sampling scenarios.

In practice, cycle token-sequences often appear multiple times with varying subsequent tokens. For instance, a poem’s title may recur across a Wikipedia page in different contexts (Section 4.3), introducing a *candidate set* of possible left-hand completions, as described in Section 3. To investigate how cycle tokens probabilistically select from a candidate set, we designed a series of experiments, building on the experimental settings of Section 4.1.

In Table 2, the Direct Stochasticity experiment expands upon the Baseline few-token experiment by introducing a candidate set $\{e2_i\}_{i=1}^n$. For each $(e1, e2_i, e3)$ combination, we generate n possible $e2_i$ tokens. For example, if $n = 3$, then a fixed $e1$ sampled from $[1, 100]$ and $e3$ from $[401, 500]$ would recur in 3 samples with 3 distinct $e2$ values

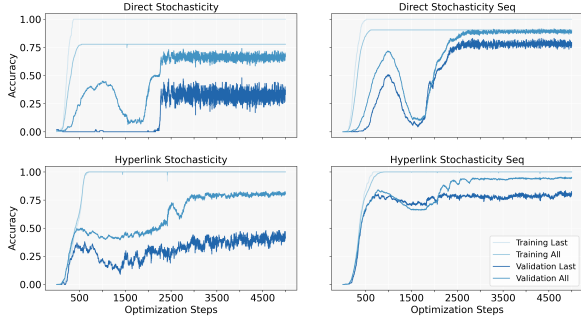


Figure 5: Accuracy curves for four stochastic few-token tasks: The left column shows token-level tasks, and the right column shows sequence-level counterparts at candidates size $n = 3$. The curves show accuracy on both the final token and all tokens, demonstrating the model’s ability to resolve specific target tokens despite stochastic variation during training.

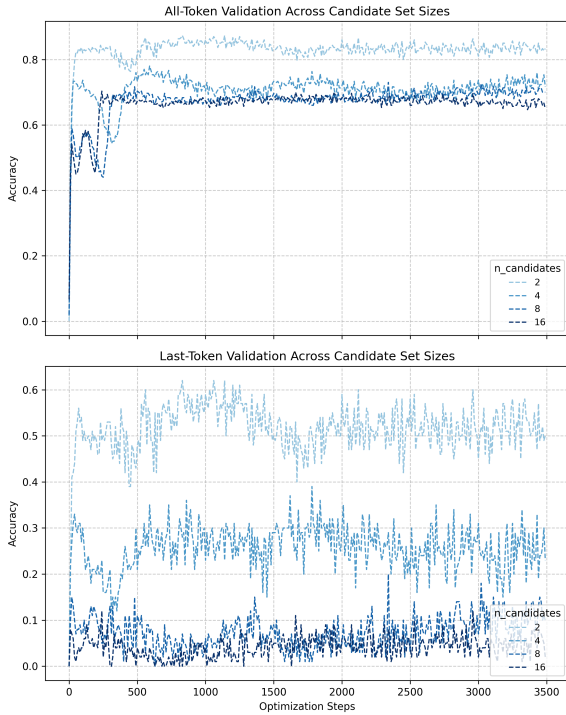


Figure 6: As the candidate set size (n) increases, the accuracy of selecting the next token following the cycle token decreases proportionally, following a $\frac{1}{n}$ pattern. This progression is not only *natural* but also *preferable* to the alternative $\frac{1}{\mathcal{V}}$, where \mathcal{V} is the vocabulary size and $\mathcal{V} \gg n$. Our RECALL-aware prompting strategy aligns with a language model’s ability to *enumerate* all n options from the candidate set.

from $[101, 400]$. The same reasoning applies to sequences, where a single $E2$ maps to n distinct sequences $\{E2_i\}_{i=1}^n$. Here, n represents the number of candidates, distinct from the sequence length \mathcal{N} .

Similarly, the Hyperlink Stochasticity

experiment expands on the Hyperlink Composability case by introducing n candidates for both $e4$ and $E4$. The cycle token continues to facilitate a “backward jump”, with additional candidates for $e5$ and $E5$ that do not disrupt the causal path.

Candidates can be targeted • Figure 5 (left) provides evidence for this, confirming that the model can access any element within the candidate set. For the single-token cases in both experiments, shown in the left-hand subplots, we set $n = 3$ and observe the model retrieving one of three $e2_i$ tokens, with an expected accuracy of $\approx 33\%$ ($\frac{1}{3}$), consistent with uniform sampling. Sequence cases, shown in the right-hand subplots, follow a similar pattern. Notably, accuracy exceeds $\frac{1}{3}$ in the sequence case, since correctly predicting the first token ensures the correctness of subsequent tokens, increasing peak validation accuracy.

Targeting is consistent across different candidate set sizes n • Specifically since n was initially fixed in our experiments, we also explored the effect of varying n . Figure 6 (left) illustrates a key observation, whereby we have an inverse relationship between the number of candidates and the accuracy of predicting the immediately succeeding token. For instance, under single-token Direct Stochasticity, 2 candidates yield $\frac{1}{2}$ accuracy, 4 candidates yield $\frac{1}{4}$, and so on.

The Reliability of The Candidate Set. *Even if the model does not select a target candidate from the candidate set, the target will still be one of the remaining $n - 1$ candidates.* Consider the case of $n = 3$. The model always achieves 33% accuracy in predicting the next token after the cycle token. That is, for each e_1 , any e_{2_i} is predicted with probability $\frac{1}{n}$. Since the candidate set contains only three possible e_{2_i} values, the only other tokens in the vocabulary to be assigned non-zero probability are the other two e_{2_i} from the candidate set. Since the model does not favor any particular e_{2_i} due to the uniform distribution of predictions, the remaining $\frac{(n-1)}{n}$, two-thirds in this example, of the *probability mass* must be split equally across the other two candidates. Otherwise, the model would retrieve target candidates with a probability of less than $\frac{1}{n}$. More generally, for any given cycle token, the model assigns non-zero probabilities to each candidate, and their probabilities are balanced. Thus, any of the n candidates will always appear within the top- n predictions, and by extension, must be included in the candidate set, even when they are not selected by the model.

Stochastic self-referencing causal cycles provide key insights into the reversal curse and tailored prompting strategies. By decoupling candidate set generation from selection, models better leverage the causal pathways created by cycle tokens for next-token prediction.

4.3 RECALL in Pretraining Corpora

4.3.1 Observing Cycle Token-Sequences

Viewing LLMs as dynamic knowledge repositories, we observe self-referencing causal cycles in widely recognized texts, which we refer to as *key writings*. These recurring phrases act as conceptual hyperlinks, guiding the retrieval of stored information. This hyperlinking behavior, while often overlooked, represents a core mechanism by which LLMs bridge long-range dependencies in text. Our key writings include timeless poems, iconic speeches, and universally familiar nursery rhymes, forming the backbone of cultural memory.

To curate our dataset, we queried ChatGPT for examples of popular texts and refined the selection to 50 key writings easily recognizable to those familiar with English literature. A detailed list, along with relevant weblinks, is provided in Appendix A. We cross-referenced this list with a Llama 3 405B model to verify the texts were part of its pretraining data. This verification was conducted offline to ensure no reliance on live queries. Additionally, we analyzed associated webpages to understand how key writings are embedded in real-world corpora (Gao et al., 2021). Examples are shown in Figures 10 and 11 (in the Appendix).

We analyzed the frequency and distribution of cycle token sequences across relevant webpages, treating each key writing’s title—or a subsequence of it—as a cycle token. These sequences act like cross-references in a library catalog, linking disparate sections of text and enabling efficient retrieval of distant information. Figure 7 illustrates how often these tokens recur, with phrases like “Star-Spangled Banner” appearing 73 times in its corresponding Wikipedia article for the U.S. anthem. Figure 8 highlights their distribution, showcasing the extensive causal pathways that cycle tokens create, allowing backward retrieval without breaking left-to-right prediction flow.

4.3.2 Utilizing RECALL

In a library, readers navigate between sections by following references or repeated titles. Similarly, cycle tokens enable language models to “jump”

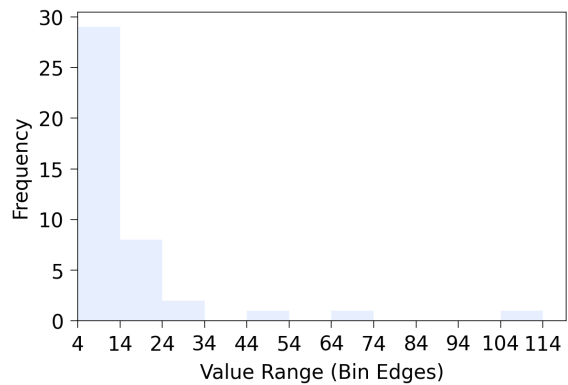


Figure 7: Frequency of cycle token-sequences across pretraining corpora for selected key writings. The bar chart illustrates the number of times cycle token-sequences, such as titles, appear as recurring motifs within corresponding webpages. Even if some occur only a few times per webpage, this suffices to create natural self-referencing causal cycles, enabling LLMs to bridge distant parts of text and achieve robust memory recall during next-token prediction.

across text sections, retrieving relevant information even when it is stored far from the original query. Without these natural hyperlinks, models often struggle to recall preceding information accurately. One example is the “preceding line problem,” adapted from (Golovneva et al., 2024). For instance, GPT-4 correctly identifies the line following “Gave proof through the night that our flag was still there” in the U.S. national anthem as “O say does that star-spangled banner yet wave.” However, when asked for the preceding line, it consistently returns incorrect responses, such as “O’er the ramparts we watch’d, were so gallantly streaming?” This issue persists across models, including Llama-3-405B (Dubey et al., 2024), Google Gemini (Team et al., 2023), and Claude 3.5 Sonnet (Anthropic, 2024), despite various prompting strategies like step-by-step reasoning, process of elimination, and structured planning.

To address this limitation, we propose a two-step RECALL-aware prompting strategy. Unlike conventional prompts, which often fail due to the next-token prediction bias, RECALL-aware prompting explicitly retrieves context before answering. This strategy mandates exploring all causal pathways, including self-referencing causal cycles, which are, strictly speaking, causal pathways, and serve as natural hyperlinks within the training data. The first step involves asking the model to recall everything it knows about the token-sequence of in-

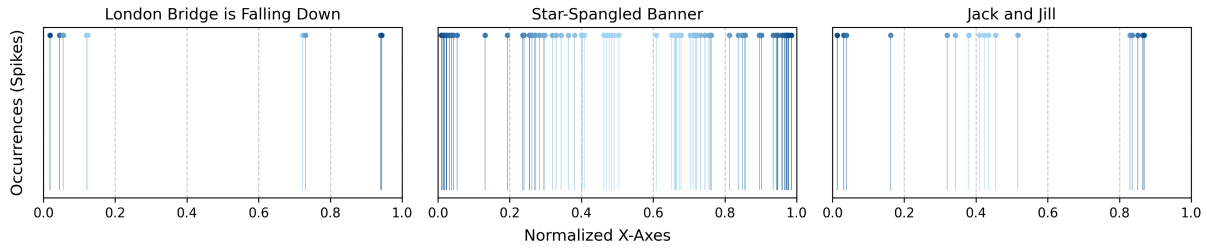


Figure 8: Spatial distribution of cycle token-sequences across key-writing texts. The lengths of each key-writing, one per subplot, are normalized to a 0-1 scale, with each vertical line representing a recurrence of the cycle token-sequence. For ease of observability, the colors at the beginning and end of the text are shaded darker blue. The visualization highlights how clusters of cycle token-sequences act as hyperlinks in pretraining data, enabling models to reference prior information effectively. Derived from the titles of key-writings, these cycle token-sequences facilitate the 'jump backward' mechanism of self-referencing causal cycles, which LLMs leverage as natural aids for robust memory retrieval.

terest. For instance, when querying the line “X” in a key-writing, we ask, “Tell me the lines surrounding this line ‘X’.” For verbose models, a broader query such as “Tell me everything you know about this line ‘X’” often suffices to retrieve the necessary context. Once the model outputs a candidate set of surrounding lines, the second step extracts the correct answer through in-context learning. This process circumvents the inherent expediency bias of next-token prediction. Figure 9 demonstrates how this approach resolves the preceding line problem in the U.S. national anthem. Vitaly, this approach is effective in 100% of our key-writings for GPT-4o (2024-12-23) and LLaMA-3.3-70B. For any given complete and intelligible sentence in the key-writing, we are able to retrieve the preceding one.

In summary, the RECALL-aware prompting process involves two steps:

1. Recollect the context: Prompt the model to provide a candidate set of answers by leveraging self-referencing causal cycles.
2. Utilize the context: Instruct the model to analyze its own outputs and extract the correct answer through in-context reasoning.

5 Conclusions

The reversal curse is a well-documented challenge in generative language models, where the model struggles to predict preceding tokens based on succeeding tokens. While prior work has primarily focused on data augmentation or architectural changes to address this limitation, we propose an alternative perspective: language models, much like libraries, may already possess latent mecha-

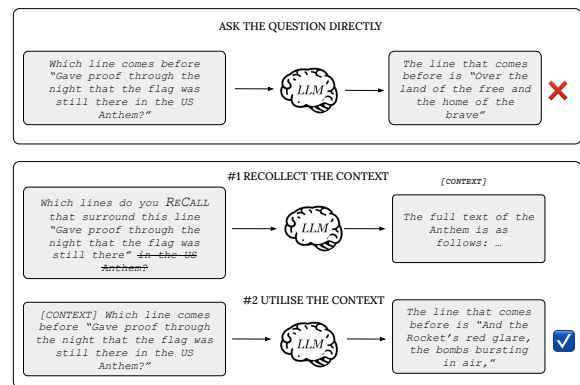


Figure 9: Two-step RECALL-aware prompting to resolve the U.S. National Anthem’s preceding line problem. The first step involves recollecting all relevant context surrounding the given line using a broad query. The second step utilizes the retrieved context to correctly identify the preceding line. This approach leverages self-referencing causal cycles to ensure the model explores all causal pathways, overcoming the reversal curse. It is effective in 100% of the key-writings we test.

nisms to cross-reference token sequences within their pretraining data.

In this work, we introduce the concept of self-referencing causal cycles (abbreviated RECALL), which act as natural hyperlinks in a model’s memory, allowing it to retrieve left-hand tokens from right-hand tokens. We demonstrate this concept at an axiomatic level through controlled few-token experiments on a small transformer model. The flexibility of these cycles under stochastic conditions suggests that RECALL mechanisms can scale to large language models. Notably, we find experimentally that self-referencing cycles emerge naturally from repeated token patterns in pretraining corpora and can enable models to overcome the reversal curse without requiring additional mod-

ifications. By leveraging these cycles, language models are able to overcome the limitations of autoregressive text generation and produce more reliable responses to prompts affected by the reversal curse.

Limitations

While this study demonstrates the potential of self-referencing causal cycles to mitigate the reversal curse, there are several limitations to consider. Our experiments are conducted in controlled settings with simplified token sequences. However, the performance of autoregressive models deployed in real-world applications may be influenced by additional factors, such as retrieval-augmented generation or web search. Further interpretability techniques may be required to precisely attribute parametric information retrieval to specific cycle tokens in the pretraining data. This poses a non-trivial challenge, as larger models often utilize fully or partially closed-source training data, and extracting pretraining data from the models themselves is intentionally designed to be difficult to maintain privacy and security.

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. 2023. Physics of language models: Part 3.2, knowledge manipulation. *arXiv preprint arXiv:2309.14402*.
- Anthropic. 2024. *Claude 3.5 Sonnet: Anthropic’s Advanced Language Model*. Accessed: 2025-01-23.
- Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. 2023. The reversal curse: LLMs trained on "a is b" fail to learn "b is a". *arXiv preprint arXiv:2309.12288*.
- Justin Chih-Yao Chen, Zifeng Wang, Hamid Palangi, Rujun Han, Sayna Ebrahimi, Long Le, Vincent Perot, Swaroop Mishra, Mohit Bansal, Chen-Yu Lee, et al. 2024. Reverse thinking makes LLMs stronger reasoners. *arXiv preprint arXiv:2411.19865*.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Leo Gao, Stella Biderman, Sidney Black, Laurence Golding, Travis Hoppe, Connor Foster, Jason Phang, Horace He, Anish Thite, Takuya Nabeshima, Samuel Presser, and Connor Leahy. 2021. *The pile: An 800gb dataset of diverse text for language modeling*. *arXiv preprint arXiv:2101.00027*.
- Olga Golovneva, Zeyuan Allen-Zhu, Jason Weston, and Sainbayar Sukhbaatar. 2024. Reverse training to nurse the reversal curse. *arXiv preprint arXiv:2403.13799*.
- Qingyan Guo, Rui Wang, Junliang Guo, Xu Tan, Jiang Bian, and Yujiu Yang. 2024. Mitigating reversal curse in large language models via semantic-aware permutation training. *CoRR*.
- Benjamin Heinzerling and Kentaro Inui. 2020. Language models as knowledge bases: On entity representations, storage capacity, and paraphrased queries. *arXiv preprint arXiv:2008.09036*.
- Harvey Lederman and Kyle Mahowald. 2024. Are language models more like libraries or like librarians? bibliotechnism, the novel reference problem, and the attitudes of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1087–1103.
- Ziming Liu, Ouail Kitouni, Niklas S Nolte, Eric Michaud, Max Tegmark, and Mike Williams. 2022. Towards understanding grokking: An effective theory of representation learning. *Advances in Neural Information Processing Systems*, 35:34651–34663.
- Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. 2019. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*.
- Jacob Mitchell Springer, Suhas Kotha, Daniel Fried, Graham Neubig, and Aditi Raghunathan. 2024. Repetition improves language model embeddings. *arXiv preprint arXiv:2402.15449*.
- Ryosuke Takahashi, Go Kamoda, Benjamin Heinzerling, Keisuke Sakaguchi, and Kentaro Inui. 2024. The curse of popularity: Popular entities have catastrophic side effects when deleting knowledge from language models. *arXiv preprint arXiv:2406.06032*.
- Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.
- A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Song Wang, Yaochen Zhu, Haochen Liu, Zaiyi Zheng, Chen Chen, and Jundong Li. 2024. Knowledge editing for large language models: A survey. *ACM Computing Surveys*, 57(3):1–37.

Da Wu, Jingye Yang, and Kai Wang. 2024. Exploring the reversal curse and other deductive logical reasoning in bert and gpt-based large language models. *Patterns*.

Appendix

A Sample Popular Texts

1. Star-Spangled Banner (U.S. Anthem)
2. London Bridge is falling down
3. Jack and Jill
4. Baa Baa Black Sheep
5. Twinkle Twinkle Little Star
6. "I have a dream" by Martin Luther King Jr.
7. Humpty Dumpty
8. Jingle Bells
9. Three Blind Mice
10. The Twelve Days of Christmas
11. Fee-fi-fo-fum
12. Three Little Kittens
13. Frosty The Snowman
14. Old Soldiers Never Die
15. Do not go gentle into that good night
16. Ain't I A Woman by Sojourner Truth
17. It's Raining, It's Pouring
18. A Dream Within A Dream
19. This Old Man
20. Peter Piper
21. Old Mother Hubbard
22. Matthew, Mark, Luke and John
23. Little Bo-Peep
24. There was an Old Woman who lived in a shoe
25. The Grand Old Duke of York
26. Solomon Grundy (nursery rhyme)
27. Star Light, Star Bright
28. John Jacob Jingleheimer Schmidt
29. Rock-a-bye Baby
30. Pussy Cat Pussy Cat
31. The Queen of Hearts
32. Barack Obama's Victory Speech 2008 (RECALL 'Yes, we can')
33. Little Miss Muffet
34. Two Little Dickie Birds
35. Little Arabella Miller
36. Little Robin Redbreast
37. Doctor Foster

38. Pat-a-cake, pat-a-cake, baker's man (RECALL Pat-a-cake)
39. Tom, Tom, the Piper's Son
40. Little Jack Horner
41. "99 Bottles of Beer" by John Donne
42. Hickory Dickory Dock

B Computing a Causally Preceding Sequence with an Autoregressive Model

B.1 The Argmax Over \mathcal{S}

We shall illustrate how we arrive at Equation 3 from Equation 1 in Section 3.

Firstly, as per Equation 1, we have:

$$S_r = \arg \max_{s \in \mathcal{S}} P_{\mathcal{M}}(s|S_l).$$

The true probability distribution of a left-hand sequence seen in the training data, given a right-hand sequence, is given by:

$$P(s|S_r) = \frac{P(S_r|s)P(s)}{P(S_r)} \quad (6)$$

as per Bayes rule. Equivalently, we have that:

$$\arg \max_{s \in \mathcal{S}} P(s|S_r) = \arg \max_{s \in \mathcal{S}} \frac{P(S_r|s)P(s)}{P(S_r)} \quad (7)$$

$$= \arg \max_{s \in \mathcal{S}} \frac{P(S_r|s)P(s)}{C} \quad (8)$$

$$= \arg \max_{s \in \mathcal{S}} P(S_r|s)P(s), \quad (9)$$

where C is a constant, indicating that $P(S_r)$ does not affect the final arg max.

If the LLM is a perfect model of the true data distribution (i.e., a perfect library),

$$\forall s \in \mathcal{S} : P(s) = P_{\mathcal{M}}(s). \quad (10)$$

Thus we have:

$$\arg \max_{s \in \mathcal{S}} P(s|S_r) = \arg \max_{s \in \mathcal{S}} P_{\mathcal{M}}(S_r|s)P_{\mathcal{M}}(s), \quad (11)$$

and finally, if S_l is indeed the most natural preceding sequence to S_r among all possible sequences, then:

$$S_l = \arg \max_{s \in \mathcal{S}} P_{\mathcal{M}}(S_r|s)P_{\mathcal{M}}(s).$$

Which is Equation 3. Here $P_{\mathcal{M}}(s)$ is given by the model as the frequency of the token-sequence s in the pre-training corpus. For rare sequences,

The Star-Spangled Banner 🌐 92 languages

Article Talk Read View source View history Tools

From Wikipedia, the free encyclopedia


For other uses, see [Star-Spangled Banner](#) (disambiguation). "Defense of Fort M'Henry" redirects here. For the 1814 battle, see [Battle of Baltimore](#).

"The Star-Spangled Banner" is the **national anthem** of the **United States**. The lyrics come from the **"Defence of Fort M'Henry"**,^[2] a poem written by American lawyer **Francis Scott Key** on September 14, 1814, after he witnessed the bombardment of **Fort McHenry** by the British **Royal Navy** during the **Battle of Baltimore** in the **War of 1812**. Key was inspired by the large **U.S. flag**, with 15 stars and 15 stripes, known as the **Star-Spangled Banner**, flying triumphantly above the fort after the battle.

The poem was set to the tune of a popular **British song** written by **John Stafford Smith** for the **Anacreontic Society**, a **social club** in **London**. Smith's song, **"To Anacreon in Heaven"** (or **"The Anacreontic Song"**), with various lyrics, was already popular in the United States. This setting, renamed **"The Star-Spangled Banner"**, soon became a popular patriotic song. With a **range** of 19 semitones, it is known for being very difficult to sing, in part because the melody sung today is the **soprano** part. Although the poem has four **stanzas**, only the first is commonly sung today with the second to fourth being rarely sung.

"The **Star-Spangled Banner**" was first recognized for official use by the **United States Navy** in 1889. On March 3, 1931,

"The Star-Spangled Banner"



The earliest surviving sheet music of "The Star-Spangled Banner" from 1814

National anthem of the United States


Lyrics [Francis Scott Key](#), 1814; 210 years ago

Music [John Stafford Smith](#), 1773; 251 years ago

Adopted March 3, 1931; 93 years ago^[1]

Preceded by ["Hail, Columbia"](#) (*de facto*) ["My Country, 'Tis of Thee"](#) (*de facto*)

Audio sample



1:19

"The Star-Spangled Banner" (instrumental version by United States Navy Band)

Figure 10: An example of a webpage containing a key-writing about the U.S. National anthem's title. The text includes self-referencing causal cycles induced by the Anthem title. This figure demonstrates how the repeated phrases within the text create natural hyperlinks, enabling the model to retrieve contextual information in the case of the "preceding line problem."

Jack and Jill 🌐 5 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia


For other uses, see [Jack and Jill](#) (disambiguation).

"Jack and Jill" (sometimes **"Jack and Gill"**, particularly in earlier versions) is a traditional English nursery rhyme. The **Roud Folk Song Index** classifies the commonest tune and its variations as number 10266,^[1] although it has been set to several others. The original rhyme dates back to the 18th century and different numbers of verses were later added, each with variations in the wording. Throughout the 19th century new versions of the story were written featuring different incidents. A number of theories continue to be advanced to explain the rhyme's historical origin.


Text [edit]

The earliest version of the rhyme was in a reprint of **John Newbery's** *Mother Goose's Melody*, thought to have been first published in London around 1765.^[2] The rhyming of "water" with "after" was taken by **Iona** and **Peter Opie** to suggest that the first verse might date from the 17th century.^[3] Jill was originally spelled Gill in the earliest version of the rhyme and the accompanying **woodcut** showed two boys at the foot of the hill.

Jack and Gill went up the hill
To fetch a pail of water;
Jack fell down and broke his crown
And Gill came tumbling after.



A postcard of the rhyme using **Dorothy M. Wheeler's** 1916 illustration ▶ Play



Mother GOOSE'S Melody. 37

JACK and GILL

Figure 11: An example of webpage containing a key-writing about the well-known poem "Jack and Jill". LLMs are able to bridge semantic gaps caused by slight variations in spelling, such as "Jack and Jill" vs. "Jack and Gill." However, this invariance enhances the robustness of self-referencing causal cycles in real-world datasets.

$P_{\mathcal{M}}(s)$ should be roughly small, and uniform, while for sequences that are popular within the dataset (see curse of popularity (Takahashi et al., 2024)), $P_{\mathcal{M}}(s)$ will have a large value.

In either case, we can use the LLM to easily compute $P_{\mathcal{M}}(s)$ as a correction factor for $P_{\mathcal{M}}(S_r|s)$, indicating an autoregressive LLM should be able to compute the most likely left-hand sequence given

a right-hand sequence, even though it was trained to give the right-hand part after the left-hand part.

B.2 The Candidate Set $S_{l_c} \subset \mathcal{S}$

One significant challenge in utilizing self-referencing causal cycles lies in efficiently computing the $\arg \max$ over possible left-hand sequences $s \in \mathcal{S}$. A brute-force approach, iterating through all candidates $s \in \mathcal{S}$, is computationally infeasible. For a fixed vocabulary size v , the total number of k -length sequences is v^k , which grows exponentially as k increases.

To address this, we aim to construct a smaller candidate set $S_{l_c} \subset \mathcal{S}$, with $|S_{l_c}| \ll |\mathcal{S}|$, that can serve as a proxy for the full set \mathcal{S} . By narrowing down the candidate set intelligently, we can perform the necessary $\arg \max$ computation efficiently. Specifically, at each iteration, we pair a candidate $s \in S_{l_c}$ with S_r and evaluate the likelihood of observing $[s, S_r]$ in context. Formally, this is expressed as:

$$S_l = \arg \max_{s \in S_{l_c}} P_{\mathcal{M}}(S_r | s) P_{\mathcal{M}}(s)$$

If S_{l_c} is constructed effectively, the $\arg \max$ computed over S_{l_c} will match the $\arg \max$ over the entire set \mathcal{S} .

C Examples are Insufficient for Reversal

We conducted a token experiment using an 8-layer transformer designed to memorize simple token-sequences of the form (e, r, f) . Each sample was constructed by randomly selecting e from $[1, 10000]$ and f from $[10001, 20000]$, pairing with one of two relation tokens: $r = 20001$ or its inverse $r' = 20002$. The notation (e, r, f) stands for entity, relation, feature, and is analogous to the (s, r, o) structure commonly used in works such as (Takahashi et al., 2024), where e and f correspond to subject and object entities, respectively. Importantly, each e is paired uniquely with an f , resulting in 20000 unique entity-feature pairings, appearing in one of four possible configurations:

$$(F, e, r, f) \text{ — (1)}$$

$$(F, f, r', e) \text{ — (2)}$$

$$(R, f, r, e) \text{ — (3)}$$

$$(R, e, r', f) \text{ — (4)}$$

For instance, e could represent names of people, f could represent names of objects, and r and

Train	(1) + 1/2 of (2)	(1) + 1/2 of (2) + (3) + 1/2 of (4)	(1) + 1/2 of (2) + 1/2 of (3)
Test	1/2' of (2)	1/2' of (2)	1/2' of (2)
Score	0	100	0
Type	Standard	Reverse Training	Generalization

Table 3: Comparison of training strategies to achieve reversal via examples.

r' could represent actions such as ‘kicks’ and ‘is kicked by,’ respectively. We also introduce special forward (F) and backward (R) directional tokens to distinguish between left-to-right and reverse-direction data. Our goal was to evaluate the notion of generalization proposed in (Golovneva et al., 2024) using a variant of this experiment, focusing on whether the model can generalize right-to-left relations from left-to-right ones.

Consider the three training configurations summarized in Table 3. In standard training, the model is shown all (e, r, f) relations and the first half of the (f, r', e) relations. It is then tested on the second half (denoted 1/2', pronounced ‘half-prime’) of the (f, r', e) relations. Here testing entails inputting the first two tokens and trying to predict the last correctly. Although the e -and- f token pairs in the second half of the (f, r', e) samples directly correspond to those in the second half of the (e, r, f) samples from training, the model struggles to infer the reverse relations, achieving a score of 0 in the second column. For instance, given ‘John kicks a ball,’ it fails to deduce ‘a ball is kicked by John.’ This is to be expected, due to the reversal curse.

In reverse training, as described in (Golovneva et al., 2024), the model is trained on all (e, r, f) relations and all (f, r, e) relations, where the latter are simply the result of applying token permutation (all-token reversal) to the former. The model is then tested on the second half of (f, r', e) relations, and achieves a perfect score of 100. This occurs because training on both (e, r, f) and (f, r, e) creates a direct causal link between f and e , making it straightforward for the model to predict e given f (even when r is substituted with r'). For example, if the model sees both ‘Mary kicks a bottle’ and ‘a bottle kicks Mary’ during training, it can easily complete ‘a bottle *is kicked by*’ with ‘Mary.’

However, reverse training does not constitute true generalization. For genuine generalization, the model must infer the second half of (f, r', e) relations without having been trained on their cor-

responding (f, r, e) relations. For instance, if the model is shown ‘John kicks a ball’ and ‘a ball is kicked by John,’ as well as ‘Mary kicks a bottle,’ it will be able to complete ‘a ball *is kicked by*’ with ‘John.’ However, will it be able to complete ‘a bottle *is kicked by*’ with ‘Mary’? This scenario is tested in the last column of Table 3, where the model fails to achieve such generalization, scoring 0. This demonstrates that under next-token prediction loss constraints, generalization of reversal is unachievable, even with examples. This is consistent with findings in prior works in the literature.

D Experimental Settings

For the deterministic few-token experiments of Section 4.1, we use a small ($\sim 90,000$ parameters) decoder-based transformer model with 2 layers and 8 attention heads. The default model embedding dimension is 36, with the exception of the Length-of-Path and candidate set size experiments, for which it is increased to 256. Training is conducted using cross-entropy loss, which is well-suited for the next-token prediction problem in transformer models. We run multiple random seeds but observe no change in results due to the well-defined nature of the problem. The experiments, implemented in PyTorch and NumPy, were performed on an NVIDIA A100 GPU and trained with the Adam optimizer (learning rate: 0.001, batch size: 1024). Reproduction requires a compute budget of slightly over 1 hour (1 hour and 9 minutes in our rerun). For the stochastic case experiments in Section 4.2, we maintain the same experimental settings, but with time required for reproduction increasing slightly to over 1.5 hours.

E Use of AI Assistants

We utilized AI assistants (e.g., ChatGPT) in paraphrasing and summarizing content from our paper to improve writing quality and precision. However, all technical contributions, experiments, and analyses were conducted by the authors without AI-generated assistance.

F Model Usage and Licensing

The language models we prompted, GPT (from OpenAI) and LLaMA (from Meta), were accessed via web interfaces under the appropriate usage terms. We ensured that our use complied with the appropriate policies of the model providers.