WHY DOES REWRITING PRETRAINING DATA IMPROVE GENERALIZATION?

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models are increasingly pretrained on synthetic rewrites of human text, which can improve downstream generalization. This gain is puzzling, as rewriting models can benefit substantially larger learners even on capabilities they do not themselves possess. We present an explanation through a rewriting-pretraining gap. We show that regularities in natural language allow rewriting models to learn simple transformations that generalize to unseen inputs. Applying these transformations changes the prediction problems encountered during pretraining, increasing the computationally bounded information accessible to the learner and inducing new representations useful for downstream tasks. We formalize this mechanism using algorithmic information theory and validate its predictions in controlled experiments. Our account further predicts that rewriting need not preserve factual content. Consistent with this prediction, we show that fictional rewrites improve compositional generalization on a challenging two-hop knowledge task.

1 INTRODUCTION

Today's large language models are pretrained on vast corpora of human-authored text (Radford et al., 2018). As computational resources grow faster than the supply of human data, model-generated synthetic data are becoming increasingly important for pretraining new models (Villalobos et al., 2024). One effective method is to use trained models to rewrite existing human data, changing its linguistic form or style while preserving its underlying content (Maini et al., 2024; Su et al., 2025; Niklaus et al., 2026). Pretraining on such data has been empirically shown to improve downstream generalization (Kang et al., 2025; Maini et al., 2025; Bai et al., 2025). This raises our key question: *Why can rewriting data with one language model improve the pretraining of another?*

Many standard explanations from machine learning fail to account for this phenomenon. One natural view is knowledge distillation (Hinton et al., 2015), in which predictive behavior of the rewriting model is transferred to the learner. However, this view does not explain why relatively small rewriting models can suffice, or even outperform larger ones, when improving substantially larger learners (Maini et al., 2025). Nor can the gains be attributed simply to task supervision, since empirically, task alignment need not be optimal, and rewritings that are stylistically unrelated to a downstream task can be more beneficial than task-aligned ones (Allen-Zhu & Li, 2023a; Niklaus et al., 2026).

Existing theories of data augmentation are likewise difficult to apply. They are typically formulated in supervised settings where augmentations preserve task labels (Dao et al., 2019; Chen et al., 2020), but in pretraining, rewriting operates on unlabeled data with no notion of a label that must remain invariant. Fundamentally, the data processing inequality (Shannon, 1948) implies that a model cannot contain more Shannon information than its data, such that learning and applying rewriting cannot add new information beyond what was already present in human data (Finzi et al., 2026). Together, these aspects make the benefits of synthetic rewriting difficult to explain and almost paradoxical.

In this paper, we develop a theoretical and empirical account of the benefits of data rewriting. Our explanation centers on a **rewriting-pretraining gap**: a rewriting transformation may be simple for a rewriting model to learn and apply, yet rewritten data can induce new nontrivial representations in the learner that improve downstream generalization. We formalize this perspective via algorithmic information theory (Li et al., 2008; Grünwald et al., 2008) and computationally bounded information (Finzi et al., 2026), and validate its predictions in controlled experiments. Our contributions are:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

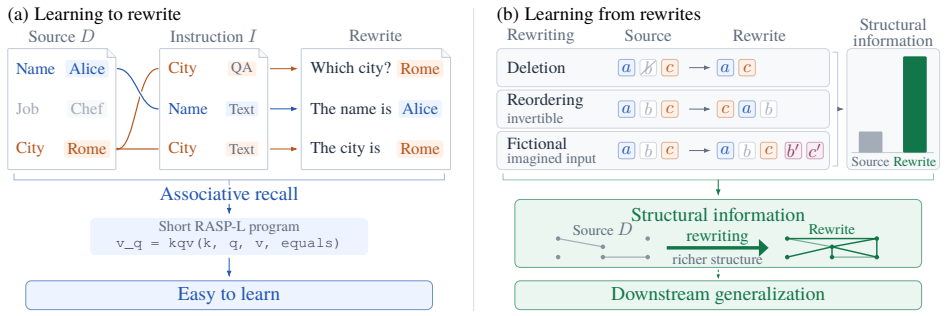

Figure 1: An illustration of rewriting-pretraining gap using a minimal model of rewriting.

- We characterize a broad class of simple rewritings as low-complexity sequential associative recall operations and provide a formalization supporting that they can be learned to generalize by transformer language models, allowing their use as rewriting models.
- We show that some such rewritings can substantially increase the computationally bounded information accessible to transformer language models pretrained on the rewritten data, by changing the encountered prediction problems and thereby inducing the learner to extract new regularities.
- Through controlled knowledge pretraining and downstream fine-tuning experiments, we show that language models trained on human data can generalize simple rewritings to unseen inputs, and that the resulting data increase computationally bounded information and downstream generalization.
- Unlike in standard approaches for synthetic data, our explanation also permits non-factual rewritings that alter the underlying content. We demonstrate that fictional rewritings of pretraining data can improve downstream generalization on a challenging two-hop knowledge composition task.

Our findings show how low-complexity rewritings can reshape pretraining to induce new regularities and improve generalization, advancing our understanding of synthetic data (Figure 1).

2 Related work

Synthetic pretraining data. As human-produced data becomes a bottleneck in pretraining language models, model-generated synthetic data is increasingly being used. An empirically successful approach is to use language models to rewrite available human data, altering its format or style while preserving its content. Prior work has studied various strategies ranging from Wikipedia-style and question-answering (QA) rephrasing (Maini et al., 2024; Su et al., 2025) to summarization (Maini et al., 2025) and pedagogical rewriting (Niklaus et al., 2026), and empirically found that pretraining on rewritten data improves downstream performance. Notably, it is known that relatively small (1–8B) rewriting models are often sufficient or better than larger (8–70B) ones (Maini et al., 2025; Kang et al., 2025; Niklaus et al., 2026), and non-QA rewriting can outperform QA rewriting on improving downstream QA tasks although stylistically less aligned (Niklaus et al., 2026; Maini et al., 2024).

Knowledge pretraining. To study generalization and memorization in language models, prior work has used controlled corpora of factual knowledge for pretraining, followed by downstream task fine-tuning and held-out evaluation (Allen-Zhu & Li, 2023a;b; Wang et al., 2024), thereby ruling out unintended exposure to evaluation data. In this setting, prior studies have identified cases where rewriting pretraining data improves downstream generalization. Namely, paraphrasing sentences of biographies improves knowledge extraction (Allen-Zhu & Li, 2023a), and adding identity statements improves reversal (Ma et al., 2026) and two-hop composition (Lin et al., 2026). Yet, each these cases were studied in isolation, and their mechanisms were attributed to data diversity or regularization. We base our work in this setting and present a unified explanation via algorithmic information theory.

3 Preliminaries

In this section, we introduce the key preliminaries in algorithmic information theory and refer to Grünwald et al. (2008) and Finzi et al. (2026) for details. All logarithms are in base two.

MDL principle. Algorithmic information theory measures the information content of data $\mathbf{x}$ as the length $K(\mathbf{x})$ of the shortest program that prints it, called its (prefix) Kolmogorov complexity (Kolmogorov, 1965; Chaitin, 1975). While this measures both structural information and incompressible

randomness, two-part descriptions separate them by having a model and then encoding the data using the model. For any computable probability p, by Shannon coding theorem, there is an encoding of data $\mathbf{x}$ of length $\lceil -\log p(\mathbf{x}) \rceil$ bits. Prepending a shortest description of p gives (Gács et al., 2001)

$$K(\mathbf{x}) \leq K(p) - \log p(\mathbf{x}) + O(1). \tag{1}$$

The minimum description length (MDL) principle (Rissanen, 2004; Grünwald, 2007) is a theoretical criterion for model selection closely related with Kolmogorov complexity. It views learning as extracting regularities from data to compress the data losslessly, and states that among a given set P of learned models, the best model $p^\star$ minimizes the two-part description length of the data,

$$p^\star = \arg\min_{p \in P} L(\mathbf{x}, p), \qquad L(\mathbf{x}, p) = |p| - \log p(\mathbf{x}), \tag{2}$$

where $|p|$ is the number of bits needed to encode a model. At equally good likelihoods (i.e. data compression), MDL favors shorter model code, embodying Occam's razor (Solomonoff, 1964).

Computationally bounded information. In the two-part codelength $L(\mathbf{x}, p)$ (2), $|p|$ measures information content that the model has learned to compress the data, and $-\log p(\mathbf{x})$ measures remaining incompressibility in the data. By limiting the model class P to a set of computationally bounded observers, the codelength $|p^\star|$ of the best model $p^\star$ can be interpreted as structural information content in the data accessible to the models in P, called epiplexity (Finzi et al., 2026). Epiplexity can increase under deterministic processing, offering a suitable notion of information for synthetic data.

Since deep learning models are overparameterized, directly encoding a model p to estimate $|p|$ ends up producing excessively large codelengths. Instead, prequential codes (Dawid, 1984) that measure the two-part codelength (2) achieve efficient codelength by avoiding separately encoding the model (Blier & Ollivier, 2018). We can then estimate model codelength by subtracting the data cost,

$$|p| = L(\mathbf{x}, p) - (-\log p(\mathbf{x})), \tag{3}$$

following prior work (Zhang et al., 2020; Whitney et al., 2020; Donoway et al., 2026; Finzi et al., 2026). We detail this procedure in practical settings in the next paragraph.

Prequential estimation. Let $P = \{p_\theta\}$ be model class and $\mathbf{x} = (\mathbf{x}_i)_{i=1}^n$ be dataset where $\mathbf{x}_i \sim p_{\text{data}}$. To measure the codelength of a learned model together with data, we imagine a communication game where a sender observes a stream of training data and must communicate them to a receiver. From a shared initialization p_{θ_0} and training algorithm A, the sender encodes each $\mathbf{x}_i$ using $p_{\theta_{i-1}}$ trained on all past data $\mathbf{x}_{1:i-1}$. The receiver reconstructs each data using a model trained on all previously reconstructed data, allowing model and data to be described without explicitly encoding the model:

$$L_{\text{preq}}(\mathbf{x}, p_\theta) = \sum_{i=1}^{n} -\log p_{\theta_{i-1}}(\mathbf{x}_i), \qquad \theta_i = A(\theta_{i-1}, \mathbf{x}_i), \qquad \theta = \theta_n. \tag{4}$$

In practice, we employ block coding, detailed in Appendix G.1. Then, to estimate learned structural information $|p_\theta|$ like in (3), we subtract the codelength of test dataset $\mathbf{y} = (\mathbf{y}_j)_{j=1}^n$ where $\mathbf{y}_j \sim p_{\text{data}}$:

$$|p_\theta| \approx |p_\theta|_{\text{preq}} \coloneqq L_{\text{preq}}(\mathbf{x}, p_\theta) - (-\log p_\theta(\mathbf{y})), \qquad -\log p_\theta(\mathbf{y}) = \sum_j -\log p_\theta(\mathbf{y}_j). \tag{5}$$

Here, using test data is crucial to avoid counting memorized sampling noise as learned information (Zhang et al., 2020). Subtracting this codelength from that of both the model and data, prequential epiplexity $|p_\theta|_{\text{preq}}$ gives an estimate of learned regularities that can be used to compresses the data.

4 Learning to Rewrite

Our central hypothesis, the rewriting-pretraining gap, consists of two parts: (i) many rewriting transformations are simple for a language model to learn and apply, and (ii) yet, rewritten data may induce novel generalizable representations in the learner language model. In this section, we formalize the first argument through the MDL principle (2). Our argument is summarized as follows:

Rewritings can be often viewed as a sequence of in-context recall followed by template filling, which are simple to represent by transformer language models and therefore easy to learn.

Table 1: Excerpts of prompts for the considered rewritings. Full details in Appendix D.

Rewriting	Prompt
Wikipedia (Maini et al., 2024)	"For the following paragraph give me a diverse paraphrase of the same in high quality English language as in sentences on Wikipedia."
Wikipedia (Kang et al., 2025)	"For the following document, regardless of its original content or formatting, write a full article of the same content in high quality English language as in texts on Wikipedia..."
QA (Kang et al., 2025)	"For the following document... convert it into a comprehensive list of question-answer pairs... where questions and answers cover complete information of the original document."
QA (Niklaus et al., 2026)	"Rewrite the document as a comprehensive FAQ... Order questions logically from foundational to advanced... Ensure the FAQ works as a standalone document."
Summarization (Niklaus et al., 2026)	"Summarize the following text. Write a standalone summary without referencing the text."

Motivation. If unrestricted, rewritings can execute arbitrary computable functions (Qiu et al., 2025), which is clearly too broad. We thus focus our analysis to rewriting on pretraining data where relatively smaller rewriting models compete with larger ones in improving downstream task performance, which constitute some of the most counterintuitive observations in this domain (Section 1). These include Wikipedia-style rewriting (Maini et al., 2024; Kang et al., 2025), QA rewriting (Kang et al., 2025; Niklaus et al., 2026), and summarization (Niklaus et al., 2026), shown in Table 1.

The rewritings share a common structure, as the source document provides most of the meaningful content, while the rewriting model changes how that content is presented. We can therefore view these transformations as in-context recall coupled with restructuring, where the model retrieves relevant information from the source and expresses it in the requested form. This aligns naturally with attention mechanism, which provides circuitry for selecting and routing information already in the context (Elhage et al., 2021; Akyürek et al., 2024). Moreover, the same transformation can be reused across documents with different subject matter, suggesting that learning to rewrite may require acquiring small circuits that broadly generalize to unseen inputs. This intuition motivates a minimal model through which we connect the representational simplicity of rewriting to its learnability.

Minimal model of rewriting. Motivated by the aforementioned properties, we model rewriting as retrieving facts from a source document in context and expressing them in prescribed forms. For this, we assume that the source is represented as a sequence of key-value pairs,

$$\mathbf{x} = (k_1, v_1, k_2, v_2, ..., k_n, v_n). \tag{6}$$

where the keys k and values v are from finite vocabularies. For example, a document about a person can be modeled as $(\texttt{name}, \texttt{Alice}, \texttt{age}, \texttt{30})$. We assume that each key appears at most once in the source, although values may repeat. We represent a rewriting instruction as a sequence

$$I = (q_1, s_1, q_2, s_2, ..., q_m, s_m), \tag{7}$$

where each q is a key present in $\mathbf{x}$, and each s is a style from a finite set, which could be for instance the declarative style of Wikipedia-style rewriting or the question-answer style of QA rewriting mentioned above. We assume that the pairs (q, s) are distinct, allowing the same key to be requested in different styles. The desired output is

$$\mathsf{rewrite}(\mathbf{x}, I) = (t_{q_1,s_1}, v_{q_1}, ..., t_{q_m,s_m}, v_{q_m}), \tag{8}$$

where v_q is the value associated with the key q, and $t_{q,s}$ denotes a template symbol that expresses the required field q and style s. We assume that keys, values, instruction, and template use disjoint vocabularies. Returning to the example document $(\texttt{name}, \texttt{Alice}, \texttt{age}, \texttt{30})$, requesting all fields in the declarative style yields "Her name is Alice. Her age is 30.", similarly as in Wikipedia-style, while QA style could yield "What is her name? Alice. How old is she? 30." In addition, requesting only selected fields abstracts the selective aspect of summarization. In every case, rewriting reuses given information while changing its selection, order, or presentation.

MDL principle and learnability. We now connect our minimal model of rewriting to the computational structure of transformers. For this, we use RASP-L (Zhou et al., 2024; Weiss et al., 2021), a programming language that expresses computations through attention mechanism and local tokenwise transformations, characterizing algorithms that transformers can represent simply. Python definitions of its core operations are provided in Listing 1 in the appendix. We now prove that:

Proposition 4.1. *The minimal model of rewriting admits a short RASP-L program (Listing 2) shared across all source documents and instruction sequences.*

A proof can be found in Appendix C.1. We now connect our minimal model of rewriting to a specific version of the MDL principle (Section 3) that predicts which functions transformers learn:

RASP MDL principle (Zhou et al., 2024, excerpt)**.** *For symbolic tasks, transformers tend to learn the shortest RASP-L program which fits the training set.*

Based on Proposition 4.1, this suggests that under sufficiently diverse training examples that rule out simpler RASP-L programs that fit the training set, transformers would learn the minimal rewriting model applicable across documents with different contents and rewriting instructions. The required diversity may arise naturally from the structure of natural language, in which shared transformations can be applied across various contents and linguistic forms (Harris, 1968; Bottou & Schölkopf, 2025). Such a program applies the same underlying operations to different source documents, selecting and restructuring information according to the rewriting instructions. Our abstraction thus provides a computational rationale for why these rewriting operations may be accessible to relatively small language models. Once the relevant linguistic capabilities are available, reusing and restructuring source information can require only modest additional computational machinery.

5 Learning from rewritten data

Having established that many rewritings can be simple for a language model to learn and apply, we now turn to the second part of our hypothesis, that these rewritings can induce novel generalizable representations in the learner language model. We formalize this argument through computationally bounded information, quantified as epiplexity (3). The key idea is the following:

> *Rewriting changes the learner's prediction problem, requiring it to capture different structure and thereby learn novel representations.*

Under our minimal model of rewriting (Section 4), we construct three rewritings that can provably increase epiplexity. This includes removing information from a source document, reordering the same information for a causal model class, and altering the factual content. The first two are reminiscent of rewritings used in practice, for instance summarization and fact-preserving rewrites such as Wikipedia and QA (Section 4). However, the last one, non-factual rewriting, is unconventional.

Setup. We consider polynomial-time computable functions $f : Z \times A \to B$ over finite sets such that, for each seed $z \in Z$, the map $f_z : A \to B$ can not be distinguished from a random function in polynomial time without knowing z, as in an encryption scheme with secret key z. For example, A and B may be names and ages, and z a hidden rule assigning facts such as $f_z(\texttt{Alice}) = \texttt{30}$. With a fixed seed z, we consider source documents $\mathbf{x}$ defined as a random variable:

$$\mathbf{x} \coloneqq (k_{\mathsf{seed}}, z, k_{\mathsf{in}}, a, k_{\mathsf{out}}, f_z(a)), \qquad a \sim \mathsf{U}(A). \tag{9}$$

We make this construction precise in Appendix B using pseudorandom functions (PRFs).

Deletion. We first consider the following rewriting instruction which deletes z:

$$I_{\mathsf{del}} = (k_{\mathsf{in}}, s, k_{\mathsf{out}}, s) \qquad \Longrightarrow \qquad \mathsf{rewrite}(\mathbf{x}, I_{\mathsf{del}}) = (t_{\mathsf{in},s}, a, t_{\mathsf{out},s}, f_z(a)), \tag{10}$$

where s is a fixed style. To compute epiplexity based on (2) and (3), we must choose a computationally bounded model class $P = \{p\}$ and a coding scheme for the model that defines $|p|$. We take P as all computable probabilistic models that support sampling and likelihood evaluation within a polynomial-time budget, and use a prefix-free coding scheme independent of a. Let $2^m \coloneqq |A|$,

$$p^\star(\mathbf{x}) = \arg\min_{p \in P} \{|p| + \mathbb{E}[-\log p(\mathbf{x} \mid m)]\} \tag{11}$$

be the model minimizing two-part MDL. Epiplexity of $\mathbf{x}$ is $|p^\star(\mathbf{x})|$ (Finzi et al., 2026). We prove,

Proposition 5.1. *Under the above setup, there exists a constant C, independent of m, such that for every sufficiently large m, some seed z satisfies*

$$|p^\star(\mathbf{x})| < C < \lfloor \log m \rfloor < |p^\star(\mathsf{rewrite}(\mathbf{x}, I_{\mathsf{del}}))|. \tag{12}$$

Therefore, for this seed z, the rewriting (10) increases epiplexity by more than $\lfloor \log m \rfloor - C$, a gap that grows without bound.

A proof can be found in Appendix C.2. Intuitively, an observed seed z lets the model predict $f_z(a)$ without learning seed-specific structure. After deleting z, encoding the shared seed-dependent rule

becomes more efficient than describing outputs individually as m grows. MDL thus favors encoding seed-specific information in the model, increasing epiplexity through its codelength.

Reordering. We now prove that, within a causal model class akin to transformer language models, even invertible rewritings that preserve all information of the source can increase epiplexity. The key idea is to employ a reordering to move the seed z to after the output $f_z(a)$, making it unavailable in the observed prefix when $f_z(a)$ is predicted.

$$I_{\mathsf{ord}} = (k_{\mathsf{in}}, s, k_{\mathsf{out}}, s, k_{\mathsf{seed}}, s) \quad \Longrightarrow \quad \mathsf{rewrite}(\mathbf{x}, I_{\mathsf{ord}}) = (t_{\mathsf{in},s}, a, t_{\mathsf{out},s}, f_z(a), t_{\mathsf{seed},s}, z), \tag{13}$$

where s is a fixed style. To compute epiplexity, we consider a model class P composed of all computable probabilistic models expressed as $p(\mathbf{x} \mid m) = \prod_{i=1}^{n} p(\mathbf{x}_i \mid \mathbf{x}_{<i}, m)$ that support sampling and likelihood evaluation within a polynomial-time budget, and prove the following:

Proposition 5.2. *Under the above setup, there exists a constant C, independent of m, such that for every sufficiently large m, some seed z satisfies*

$$|p^\star(\mathbf{x})| < C < \lfloor \log m \rfloor < |p^\star(\mathsf{rewrite}(\mathbf{x}, I_{\mathsf{ord}}))|. \tag{14}$$

Hence a fixed invertible rewriting can increase causal epiplexity by an unbounded amount.

A proof can be found in Appendix C.3. When the seed z precedes $f_z(a)$, the model can predict it without encoding the seed. Reordering makes z inaccessible during causal prediction, favoring its encoding in the model and increasing epiplexity, as with deletion.

Fictional rewriting. While the aforementioned rewrites are close to ones used in practice, we now show that a less conventional choice of non-factual rewritings that alter the underlying facts of the source document can also increase epiplexity.

Let us define a *fictional rewriting* as a rewriting that keeps the seed z disclosed and the rule f_z fixed, but depicts a new, "imagined" input $a' \neq a$ in place of the existing input a, together with the updated output $f_z(a')$. To choose the imagined input, we use a polynomial-time computable function $\pi : W \times A \to A$ over finite sets, such that for each seed $w \in W$ the map $\pi_w : A \to A$ is a permutation of A. In analogy with f, we assume that π_w cannot be distinguished from a random permutation in polynomial time if w is not known. We additionally assume that w is fixed across the data distribution and $|W| \leq 2^{m/3}$, so that w is shorter than an input. This seed is different from z, and is never written in the document. With these, the new input is $a' = \pi_w(a)$ and its output is $f_z(\pi_w(a))$, computed under the disclosed seed.

We compare two ways of extending the source $\mathbf{x}$ using additional pair of template and value: fictional and factual. The fictional instruction add a template $t_{\mathsf{out,fic}}$ and apply it to $f_z(\pi_w(a))$:

$$\begin{aligned} I_{\mathsf{fic}} &= (k_{\mathsf{seed}}, s, k_{\mathsf{in}}, s, k_{\mathsf{out}}, s, k_{\mathsf{in}}, s_{\mathsf{fic}}, k_{\mathsf{out}}, s_{\mathsf{fic}}) \\ \Longrightarrow \quad \mathsf{rewrite}(\mathbf{x}, I_{\mathsf{fic}}) &= (t_{\mathsf{seed},s}, z, t_{\mathsf{in},s}, a, t_{\mathsf{out},s}, f_z(a), t_{\mathsf{in,fic}}, \pi_w(a), t_{\mathsf{out,fic}}, f_z(\pi_w(a))). \end{aligned} \tag{15}$$

The factual instruction add a template $t_{\mathsf{out},s'}$ and apply it to $f_z(a)$, while leaving a unpermuted.

$$\begin{aligned} I_{\mathsf{fact}} &= (k_{\mathsf{seed}}, s, k_{\mathsf{in}}, s, k_{\mathsf{out}}, s, k_{\mathsf{in}}, s',\ k_{\mathsf{out}}, s') \\ \Longrightarrow \quad \mathsf{rewrite}(\mathbf{x}, I_{\mathsf{fact}}) &= (t_{\mathsf{seed},s}, z, t_{\mathsf{in},s}, a, t_{\mathsf{out},s}, f_z(a), t_{\mathsf{in},s'}, a,\ t_{\mathsf{out},s'}, f_z(a)). \end{aligned} \tag{16}$$

Proposition 5.3. *Under the above setup, there exists a constant C, independent of m, such that for every sufficiently large m, some pair of seeds (z, w) satisfies*

$$\max\{|p^\star(\mathbf{x})|,\ |p^\star(\mathsf{rewrite}(\mathbf{x}, I_{\mathsf{fact}}))|\} < C < \lfloor \log m \rfloor < |p^\star(\mathsf{rewrite}(\mathbf{x}, I_{\mathsf{fic}}))| \tag{17}$$

Thus a fictional rewriting can increase epiplexity more than a factual rewriting.

A proof can be found in Appendix C.4. Intuitively, the factual extension adds no unexplained structure because a and $f_z(a)$ are already determined by the known z and a. In contrast, the fictional extension introduces $\pi_w(a)$, whose shared regularity depends on an undisclosed seed w. As the dataset grows, encoding w becomes cheaper than describing the transformed input individually, so minimizing description length favors learning this rule and increases epiplexity.

Together, these constructions show that rewriting can substantially change the structure available to a learner. Removing information or changing its order can make the rewritten data require more structure to predict, even when no new information is added. The fictional construction shows that this effect does not require preserving the factual content of the source. Thus, rewriting does not need to add new knowledge or perform only a fact-preserving transformation to be useful. Simply changing how information is presented can change the prediction problem and lead the learner to capture new structure, which can in turn support better generalization.

6 EXPERIMENTS

Through our experiments, we aim to answer the following key questions:

Q1. Can a language model trained on natural language perform rewritings on unseen inputs, although it cannot solve the downstream task those rewritings improve? (Section 6.1)

Q2. When pretraining on rewritten data improves generalization, does this coincide with an increase in structural information compressed in the learner? (Section 6.2)

Q3. Can fictional rewritings that do not preserve factuality improve generalization on a downstream task by increasing structural information? (Section 6.3)

Setup. Following prior work on generalization and memorization in language models (Section 2), we employ controlled knowledge corpora. We split the basic facts D into "in-distribution" (ID) and "out-of-distribution" (OOD) sets D_{ID} and D_{OOD}, and apply task-specific deduction rules within each split to construct the respective inferred facts F_{ID} and F_{OOD}. Models are pretrained on D or its rewritten version D', fine-tuned on F_{ID}, and evaluated on F_{OOD} to test rule generalization. Here, rewriting is applied only to the ID facts, therefore, D' contains both D and rewritten D_{ID}. We consider problems with known useful rewrites, described below. Examples are in Appendix F.

- **Knowledge extraction.** This task asks whether a learner can answer questions about people after pretraining on their biographies. Following Allen-Zhu & Li (2023a), D has synthetic biographies of 200,000 distinct people, split equally into ID and OOD. Each biography has six sentences stating their birth date, birth city, university, major, employer, and work city, in that order. For the downstream task, each inferred fact gives a person and asks for an attribute. It is known that models pretrained on D fail to associate each person with all attributes, leading to poor generalization. A known useful rewriting adds paraphrases of each biography (Allen-Zhu & Li, 2023a).
- **Knowledge reversal.** Also called inverse search, this task asks whether a learner can make deductions of the form "B is A" after pretraining on "A is B". Following Ma et al. (2026), we build D as a dataset of 1,000 facts "h has a husband named t", each with a unique head h and tail t, split into 80% D_{ID} and 20% D_{OOD}. For the downstream task, each inferred fact provides a tail t and asks for the related head h. Language models are known to struggle to generalize, a phenomenon known as the reversal curse (Berglund et al., 2023). A known helpful rewriting creates an identity fact "e is e" for each head and tail entity in D_{ID}, called the identity bridge (Ma et al., 2026).
- **Knowledge composition.** This task tests whether a learner can perform two-hop deductions such as "A's dad's age is C" after pretraining on "A's dad is B" and "B's age is C". Following Lin et al. (2026), D contains (h, r_1, b) or (b, r_2, t), where h, b, t are head, bridge, and tail entities from disjoint sets H, B, T each of size 500, and r_1 and r_2 are relations from disjoint sets R_1, R_2 each of size five. We split each hop's facts 80% ID and 20% OOD. For the downstream task, each inferred fact in F_{ID} provides (h, r_1, r_2) and asks for t, based on $(h, r_1, b), (b, r_2, t) \in D_{\mathsf{ID}}$, with the same applying to F_{OOD}. This task requires challenging compositional generalization (Wang et al., 2024). A known helpful rewriting creates an identity fact (b, r_{id}, b) for each bridge $b \in B$.

Throughout our experiments, we use standard decoder-only transformer models with 1.1B parameters, leveraging TinyLlama (Zhang et al., 2024) architecture and, optionally, trained weights. For the rewriting model, we use TinyLlama-1.1B-Chat which is pretrained on raw human text and then tuned for chat, as our rewrites are specified by an instruction. The learner model is pretrained from scratch followed by fine-tuning, allowing us to isolate the effect of data rewriting during pretraining.

6.1 LANGUAGE MODELS CAN REWRITE FOR TASKS THEY CANNOT SOLVE (Q1)

To answer **Q1**, we test whether TinyLlama-1.1B-Chat can successfully rewrite unseen facts in D to help a downstream learner solve a task, and whether it itself generalizes to the task after continued pretraining on D and fine-tuning on F_{ID}. To quantify the accuracy of rewriting, we measure exact-match accuracy for reversal and composition, where each rewrite has a unique correct output. For extraction, since there is no unique paraphrasing, we score a rewritten biography as correct if it preserves all six attribute values. We use rejection sampling to suppress noisy rewrites (Appendix E).

The results are in Table 2. The rewriting model almost perfectly produces identity rewrites for knowledge reversal and two-hop composition. The few failures come from entity names that the

Table 2: Rewriting accuracy (%) of the rewriting model on unseen basic facts in D versus its own test accuracy (%) on F_{OOD} after continued pretraining on D and fine-tuning on F_{ID}.

Task	Rewriting	RW Acc.	Task Acc.
Extraction	• *Olivia Mckenzie Johns arrived on December 3, 1834. He feels a deep connection to …* → *Olivia Mckenzie Johns was born on December 3, 1834, in Baton Rouge, Louisiana, …*	91.6	6.3
Reversal	• *Sinclair has a husband named Paul.* (h, r, t) → *Paul has a self named Paul.* (t, r_{id}, t)	100.0	0.0
Composition	• *Caldman* (b) → *Caldman has a self named Caldman.* (b, r_{id}, b) • *Mereden has a spouse named Anthony.* (h, r_1, b) → *Mereden has a spouse whose self is named Anthony.* $(h, r_1, r_{\mathrm{id}}, b)$ • *Anthony has a boss named Pemwick.* (b, r_2, t) → *Anthony has a self whose boss is named Pemwick.* $(b, r_{\mathrm{id}}, r_2, t)$	100.0	0.7

Table 3: Downstream OOD accuracy and estimated epiplexity across three randomized experiments.

Task	Rewriting	OOD Accuracy (%) first-token	OOD Accuracy (%) exact	Epiplexity (Mbit)
Extraction	×	19.7 ± 1.4	4.5 ± 0.7	60.947 ± 2.468
	○	$\mathbf{34.5 \pm 1.3}$	$\mathbf{27.5 \pm 2.9}$	$\mathbf{176.710 \pm 0.973}$
Reversal	×	0.7 ± 0.3	0.0 ± 0.0	0.094 ± 0.001
	○	$\mathbf{24.3 \pm 5.8}$	$\mathbf{23.3 \pm 6}$	$\mathbf{0.113 \pm 0.003}$
Composition	×	4.2 ± 0.6	2.8 ± 0.3	0.287 ± 0.007
	○	$\mathbf{16.0 \pm 4.8}$	$\mathbf{13.9 \pm 4.9}$	$\mathbf{0.291 \pm 0.005}$

model occasionally misspells, which is mitigated by using one more sample (Appendix E). Likewise, for extraction, the model performs a correct rewrite for 91.6% of all biographies. Notably, by contrast, its own downstream test accuracy after continued training is near zero across all tasks. The model thus successfully produces rewrites for a task without being able to solve that task itself, thereby answering **Q1** affirmatively. Next subsection uses rewritten datasets built in this subsection.

6.2 Rewriting improves downstream generalization and epiplexity (Q2)

To answer **Q2**, we pretrain randomly initialized TinyLlama-1.1B models on original (D) or rewritten (D') basic facts, supplemented with identical deductions from D_{ID}. We then fine-tune them on ID inferred facts F_{ID}, and evaluate them on OOD inferred facts F_{OOD}. To test whether the resulting improvement in generalization coincides with an increase in the structural information compressed in the learner, we estimate and compare epiplexity (5) between the original and rewritten pretraining data, using blockwise prequential estimation (Perez et al., 2021) which fits our token-matched multi-epoch setting. Details are provided in Appendix G.1.

The results are in Table 3, showing that rewriting pretraining data consistently improves downstream OOD generalization. This is consistent with, and extends, prior findings on those knowledge tasks (Allen-Zhu & Li, 2023a; Ma et al., 2026; Lin et al., 2026). Furthermore, the estimated epiplexity increases in all tested cases, supporting that improved generalization through simple rewritings coincide with an increase in structural regularities, thereby providing an affirmative answer to **Q2**.

6.3 Fictional rewriting can improve generalization and epiplexity (Q3)

While all rewritings in the previous experiments preserve facts, as per Section 5, our explanation predicts that a rewriting need not be factual to improve downstream generalization. It only needs to create new structural information across pretraining data which is useful for downstream generalization. To validate this hypothesis (**Q3**), we test a novel non-factual, or fictional, rewriting.

To this end, we take the basic facts in D to define the world. A rewrite is factual if every statement it produces is entailed by the facts under this world. Rephrasing and identity rewrites are factual in this sense. A rewrite is fictional if it introduces a relation absent from the world and use them arbitrarily, so that its statements are not entailed by the basic facts (Proposition 5.3). Since downstream tasks ask about the world, any gain from fictional data cannot come from the facts it conveys.

On the knowledge composition task, we propose a fictional rewrite which we call *Bichsel's Revenge*. It is inspired by "A Table Is a Table", a short story by the writer Peter Bichsel (Bichsel, 1969). In

Table 4: Rewriting accuracy (%) of the rewriting model on unseen facts in D versus its own test accuracy (%) on F_{OOD} after continued pretraining on D and fine-tuning on F_{ID} of composition task.

Rewriting	**RW Acc.**	**Task Acc.**
Anthony has a boss named Pemwick. (b, r_2, t)		
$\rightarrow$ *Anthony has a giver named Larkbury.* $(b, r_\pi, \pi(b))$	100	
$\rightarrow$ *Larkbury has a receiver named Anthony.* $(\pi(b), r_\pi^{-1}, b)$	100	
$\rightarrow$ *Anthony has a giver whose receiver is named Anthony.* $(b, r_\pi, r_\pi^{-1}, b)$	100	0.7
$\rightarrow$ *Larkbury has a receiver whose giver is named Larkbury.* $(\pi(b), r_\pi^{-1}, r_\pi, \pi(b))$	100	
$\rightarrow$ *Mereden has a spouse whose giver is named Larkbury.* $(h, r_1, r_\pi, \pi(b))$	100	
$\rightarrow$ *Larkbury has a receiver whose boss is named Pemwick.* $(\pi(b), r_\pi^{-1}, r_2, t)$	100	

Table 5: Downstream OOD accuracy and estimated epiplexity across three randomized experiments.

Task	**Rewriting**	**OOD Accuracy** (%) first-token	**OOD Accuracy** (%) exact	**Epiplexity** (Mbit)
Composition	$\times$	4.2 ± 0.6	2.8 ± 0.3	0.287 ± 0.007
	Identity	16.0 ± 4.8	13.9 ± 4.9	0.291 ± 0.005
	Fictional (Bichsel's Revenge)	$\mathbf{24.3 \pm 1.9}$	$\mathbf{22.1 \pm 2.2}$	$\mathbf{0.385 \pm 0.002}$

the story, a lonely old man is bored with a life where nothing ever changes. He decides to shake things up by renaming everything around him: his bed becomes a "picture", the table becomes a "carpet", the chair an "alarm clock", and so on. He practices his new vocabulary until he forgets the original words. In the end he can no longer understand other people, they can't understand him, and he stops speaking altogether, left completely isolated. While this tragedy is often used to illustrate how private language loses communication with language as a social object (Wittgenstein, 1968), we take an opposite view and use renaming to create a rewriting that increases structural information.

We define Bichsel's Revenge as follows. Let $\pi : B \to B$ be a fixed renaming of the bridge entities B. We introduce two new relations, r_π and r_π^{-1}, defined by $r_\pi : b \mapsto \pi(b)$ and $r_\pi^{-1} : \pi(b) \mapsto b$. The rewritten data then contain, for every $b \in B$, basic facts $(b, r_\pi, \pi(b))$ and $(\pi(b), r_\pi^{-1}, b)$, the round-trip facts $(b, r_\pi, r_\pi^{-1}, b)$ and $(\pi(b), r_\pi^{-1}, r_\pi, \pi(b))$, together with the two-hop facts $(h, r_1, r_\pi, \pi(b))$ and $(\pi(b), r_\pi^{-1}, r_2, t)$ for every ID basic fact (h, r_1, b) and (b, r_2, t). As such, this rewriting introduces relations implied by an arbitrary choice of renaming nonexistent in the world.

We test this rewriting in Tables 4 and 5. Given π, the rewriting model almost perfectly rewrites unseen inputs, indicating simplicity (also see Appendix F). Yet, we find that it substantially improves compositional generalization compared to the identity bridge rewriting proposed in prior work (Lin et al., 2026), with a larger epiplexity increase suggesting more learned structure. Notably, since $\pi = \mathrm{id}$ reduces the definition to identity bridge, we can see that the use of nontrivial renaming is important.

Following Lin et al. (2026), we test whether fictional rewriting makes first-hop states more useful for second-hop prediction as in identity rewriting (details in Appendix G.3). We extract the bridge hidden state from one task query (source) and use it to replace the corresponding state in another query (target) with the same relations but a different head. Figure 2 shows the fraction of pairs where the patched model prefers the source query's answer over the target's. This preference in OOD compositions increases under fictional rewrites, indicating more generalizable representation (usable first-hop states).

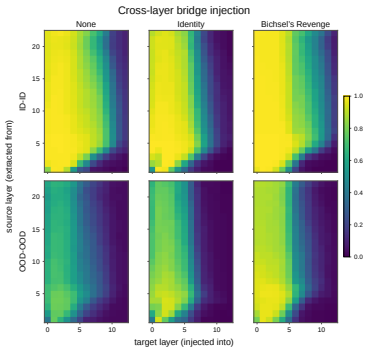

Figure 2: A visualization of cross-layer transfer of first-hop states.

7 Conclusion

We propose the rewriting-pretraining gap to explain how one model can improve another's generalization by rewriting pretraining data without possessing the target capabilities itself. Simple rewritings have short description lengths and are learnable from linguistic regularities, whereas learning to predict their outputs encourages the learner to capture additional structure that supports downstream generalization. Our experiments show that rewriting can improve generalization even without preserving factuality. A limitation is that whether these effects extend to large-scale pretraining is left for future work, which requires more care to prevent data leak. Our findings suggest designing synthetic data for the structure they induce in learners, instead of solely for the content they preserve.

AI USE STATEMENT

We used generative AI tools to generate names for the synthetic datasets introduced in Section 6, assist in developing and refining initial ideas for the theoretical framework and propositions, identify potentially relevant literature, and polish grammar. No contents were directly adopted without verification. We fully assume responsibility for the content of this paper.

ETHICS STATEMENT

Our work introduces theoretical framework for analyzing data rewriting and experiments to support our hypothesis. Our experiments use clearly defined synthetic data and publicly available language models. The study does not involve private or sensitive personal data. We are not aware of any ethical concerns arising from the methods or experiments presented in this work.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of datasets, rewriting procedures, model, evaluation protocols, and generation prompts in main text and the appendix. The appendix additionally reports the hyperparameters and training details needed to reproduce our results.

REFERENCES

Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. In-context language learning: Architectures and algorithms. *arXiv preprint arXiv:2401.12973*, 2024. (page 4)

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction. *arXiv preprint arXiv:2309.14316*, 2023a. (pages 1, 2, 7, 8, 22, 28)

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.2, knowledge manipulation. *arXiv preprint arXiv:2309.14402*, 2023b. (page 2)

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025. (page 1)

Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. The reversal curse: Llms trained on" a is b" fail to learn" b is a". *arXiv preprint arXiv:2309.12288*, 2023. (page 7)

Peter Bichsel. Kindergeschichten, 1969. (page 8)

Eden Biran, Daniela Gottesman, Sohee Yang, Mor Geva, and Amir Globerson. 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*, pp. 14113–14130, 2024. (page 29)

Léonard Blier and Yann Ollivier. The description length of deep learning models. *Advances in Neural Information Processing Systems*, 31, 2018. (page 3)

Léon Bottou and Bernhard Schölkopf. The fiction machine. *SIAM News*, 58(3), 2025. (page 5)

Gregory J Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM (JACM)*, 22(3):329–340, 1975. (page 2)

Shuxiao Chen, Edgar Dobriban, and Jane H Lee. A group-theoretic framework for data augmentation. *Journal of Machine Learning Research*, 21(245):1–71, 2020. (page 1)

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

Tri Dao, Albert Gu, Alexander Ratner, Virginia Smith, Chris De Sa, and Christopher Ré. A kernel theory of modern data augmentation. In *International conference on machine learning*, pp. 1528–1537. PMLR, 2019. (page 1)

A Philip Dawid. Present position and potential developments: Some personal views statistical theory the prequential approach. *Journal of the Royal Statistical Society: Series A (General)*, 147(2): 278–290, 1984. (page 3)

Elizabeth Donoway, Hailey Joren, Fabien Roger, and Jan Leike. Excess description length of learning generalizable predictors. *arXiv preprint arXiv:2601.04728*, 2026. (page 3)

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1(1):12, 2021. (page 4)

Marc Finzi, Shikai Qiu, Yiding Jiang, Pavel Izmailov, J Zico Kolter, and Andrew Gordon Wilson. From entropy to epiplexity: Rethinking information for computationally bounded intelligence. *arXiv preprint arXiv:2601.03220*, 2026. (pages 1, 2, 3, 5, 14, 18)

Péter Gács, John T Tromp, and Paul MB Vitányi. Algorithmic statistics. *IEEE Transactions on Information Theory*, 47(6):2443–2463, 2001. (page 3)

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986. doi: 10.1145/6490.6503. URL https://doi.org/10.1145/6490.6503. (page 14)

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Peter D Grünwald. *The minimum description length principle*. MIT press, 2007. (page 3)

Peter D Grünwald, Paul MB Vitányi, et al. Algorithmic information theory. *Handbook of the Philosophy of Information*, pp. 281–320, 2008. (pages 1, 2)

Zellig Harris. Mathematical structures of language. *Interscience tracts in pure and applied mathematics*, 1968. (page 5)

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. (page 1)

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Feiyang Kang, Newsha Ardalani, Michael Kuchnik, Youssef Emad, Mostafa Elhoushi, Shubhabrata Sengupta, Shang-Wen Li, Ramya Raghavendra, Ruoxi Jia, and Carole-Jean Wu. Demystifying synthetic data in llm pre-training: A systematic study of scaling laws, benefits, and pitfalls. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10750–10769, 2025. (pages 1, 2, 4, 20, 21)

Andrei N Kolmogorov. Three approaches to the quantitative definition ofinformation'. *Problems of information transmission*, 1(1):1–7, 1965. (page 2)

Ming Li, Paul Vitányi, and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008. (page 1)

Pengxiao Lin, Zheng-An Chen, and Zhi-Qin John Xu. Unveiling the mechanisms of multi-hop reasoning in transformers via identity bridge. In *Third Conference on Language Modeling*, 2026. (pages 2, 7, 8, 9)

Xutao Ma, Yixiao Huang, Hanlin Zhu, and Somayeh Sojoudi. Breaking the reversal curse in autoregressive language models via identity bridge. *arXiv preprint arXiv:2602.02470*, 2026. (pages 2, 7, 8)

Pratyush Maini, Skyler Seto, Richard Bai, David Grangier, Yizhe Zhang, and Navdeep Jaitly. Rephrasing the web: A recipe for compute and data-efficient language modeling. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14044–14072, 2024. (pages 1, 2, 4, 20, 21)

Pratyush Maini, Vineeth Dorna, Parth Doshi, Aldo Carranza, Fan Pan, Jack Urbanek, Paul Burstein, Alex Fang, Alvin Deng, Amro Abbas, et al. Beyondweb: Lessons from scaling synthetic data for trillion-scale pretraining. *arXiv preprint arXiv:2508.10975*, 2025. (pages 1, 2, 20)

Thao Nguyen, Yang Li, Olga Golovneva, Luke Zettlemoyer, Sewoong Oh, Ludwig Schmidt, and Xian Li. Recycling the web: A method to enhance pre-training data quality and quantity for language models. *arXiv preprint arXiv:2506.04689*, 2025. (page 20)

Joel Niklaus, Atsuki Yamaguchi, Michal Štefánik, Guilherme Penedo, Hynek Kydlíček, Elie Bakouch, Lewis Tunstall, Edward Emanuel Beeching, Thibaud Frere, Colin Raffel, et al. How can we synthesize high-quality pretraining data? a systematic study of prompt design, generator model, and source data. *arXiv preprint arXiv:2604.13977*, 2026. (pages 1, 2, 4, 20, 21)

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. Rissanen data analysis: Examining dataset characteristics via description length. In *International Conference on Machine Learning*, pp. 8500–8513. PMLR, 2021. (page 8)

Ruizhong Qiu, Zhe Xu, Wenxuan Bao, and Hanghang Tong. Ask, and it shall be given: On the turing completeness of prompting. In *International Conference on Learning Representations*, volume 2025, pp. 6286–6309, 2025. (page 4)

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training, 2018. (page 1)

Jorma Rissanen. Minimum description length principle. *Encyclopedia of statistical sciences*, 7, 2004. (page 3)

Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948. (page 1)

Ray J Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1): 1–22, 1964. (page 3)

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2459–2475, 2025. (pages 1, 2, 20)

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Position: Will we run out of data? limits of llm scaling based on human-generated data. In *Forty-first International Conference on Machine Learning*, 2024. (page 1)

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. *arXiv preprint arXiv:2405.15071*, 2024. (pages 2, 7, 29)

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *International Conference on Machine Learning*, pp. 11080–11090. PMLR, 2021. (page 4)

William F Whitney, Min Jae Song, David Brandfonbrener, Jaan Altosaar, and Kyunghyun Cho. Evaluating representations by the complexity of learning low-loss predictors. *arXiv preprint arXiv:2009.07368*, 2020. (page 3)

Ludwig Wittgenstein. *Philosophical investigations. Wittgenstein*. Macmillan, 1968. (page 9)

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024. (page 7)

Xiao Zhang, Xingjian Li, Dejing Dou, and Ji Wu. Measuring information transfer in neural networks. *arXiv preprint arXiv:2009.07624*, 2020. (page 3)

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Joshua Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. In *International Conference on Learning Representations*, volume 2024, pp. 15898–15926, 2024. (pages 4, 5, 14)

A RASP-L CODE

Here, we provide a core and library RASP-L functions from Zhou et al. (2024). We also provide a short RASP-L program for our minimal rewriting model, using these functions.

```python
def tok_map(x, f):
    return np.array(
        [f(a) for a in x], dtype=int)

def seq_map(x, y, f):
    return np.array(
        [f(a, b) for a, b in zip(x, y)],
        dtype=int)

def indices(x):
    return np.arange(len(x))
```

```python
def kqv(k, q, v, pred, default=0):
    out = np.full(len(q), default, dtype=int)
    for i in range(len(q)):
        selected = [
            v[j] for j in range(i + 1)
            if pred(k[j], q[i])]
        if selected:
            out[i] = int(np.mean(selected))
    return out

def shift_right(x, n, default=0):
    return kqv(
        indices(x) + n, indices(x), x,
        lambda a, b: a == b,
        default=default)
```

Listing 1: Numpy implementation of RASP-L core and library functions.

```python
# Fixed tokenwise functions.
def value_at(curr, prev):
    if curr == INS: return INS
    if curr == OUT: return EOS
    if curr in V: return curr
    if prev in K and curr in S:
        return TEMPLATE[prev, curr]
    return NULL

def query_at(curr, prev):
    if curr == OUT: return INS
    if curr in FIELD: return FIELD[curr]
    if curr in V and prev in FIELD: return prev
    return NOQUERY

def merge(a, b):
    return b if a == NULL else a
```

```python
# Straight-line RASP-L program.
def rewrite(x):
    prev = shift_right(x, 1, default=NULL)
    values = seq_map(x, prev, value_at)
    back1 = shift_right(values, 1, default=NULL)
    back2 = shift_right(values, 2, default=NULL)

    # Keys for source, instruction, and termination.
    source = seq_map(x, prev,
        lambda a, b: b if a in V and b in K else NULL)
    plan = seq_map(x, back2,
        lambda a, b: b if a in S else NULL)
    end = seq_map(x, back1,
        lambda a, b: b if a == OUT else NULL)

    # Combine the entries and perform lookup.
    keys = seq_map(source, plan, merge)
    keys = seq_map(keys, end, merge)
    queries = seq_map(x, prev, query_at)
    return kqv(keys, queries, values,
               lambda a, b: a == b, default=NULL)
```

Listing 2: A RASP-L program implementing the minimal rewriting model.

B PSEUDORANDOM FUNCTIONS

We index all objects by $m \in \mathbb{N}$ and take $|Z| = |A| = 2^m$, $|B| = 2^{4m}$, identifying elements with binary strings. A family $f_z : A \to B$ is a *pseudorandom function* (PRF) if (i) a single algorithm, independent of m, computes $f_z(a)$ from (z, a) for $a \in A$ in $\mathrm{poly}(m)$ time, and (ii) for every nonuniform probabilistic polynomial-time oracle algorithm $\mathcal{A}$,

$$\left| \Pr_{z \sim \mathsf{U}(Z)} \left[\mathcal{A}^{f_z}(1^m) = 1 \right] - \Pr_{R \sim \mathsf{U}(B^A)} \left[\mathcal{A}^{R}(1^m) = 1 \right] \right| \leq \varepsilon(m) \tag{18}$$

for some negligible ε. PRFs exist if one-way functions exist (Goldreich et al., 1986).

The time budget of P is a polynomial large enough to evaluate f, and models are coded by a globally prefix-free code independent of m, under which hard-coding z costs $m + O(\log m)$ bits. By (i), a model evaluating f_z on an observed seed has constant length. By (ii), every $p \in P$ satisfies (Finzi et al., 2026, Appendix A.4, Eq. (32))

$$\Pr_z \Big[\mathbb{E}_a \big[-\log p(a, f_z(a) \mid m) \big] \leq C \Big] \leq 2\varepsilon(m) + 2^{1+2C-5m}, \tag{19}$$

where C is a constant and nonuniformity lets ε be uniform over models of length at most $\lfloor \log m \rfloor$. The choice $|B| = 2^{4m}$ makes the second term vanish for $C = 2m + O(\log m)$.

C PROOF

C.1 PROOF OF PROPOSITION 4.1

Proof. Let

$$P = (D, \langle \mathsf{ins} \rangle, q_1, s_1, \ldots, q_m, s_m, \langle \mathsf{out} \rangle) \tag{20}$$

be the serialized prompt. For each request (q_j, s_j), write T_j for its template token t_{q_j,s_j}. We extend this notation with the instruction-start and output-end symbols:

$$T_j = t_{q_j,s_j} \quad (1 \le j \le m), \tag{21}$$

$$T_0 = \langle \mathsf{ins} \rangle, \tag{22}$$

$$T_{m+1} = \langle \mathsf{eos} \rangle. \tag{23}$$

Thus the source lookup and the transitions between successive requests can be written as

$$\mathcal{L} = \{k_i \mapsto v_i : 1 \le i \le n\} \cup \{T_j \mapsto T_{j+1} : 0 \le j \le m\}. \tag{24}$$

The second set includes the transition to the first template and the transition from the last template to the end symbol. When $m = 0$, it consists only of $\langle \mathsf{ins} \rangle \mapsto \langle \mathsf{eos} \rangle$.

Let $Y = \mathsf{rewrite}(D, I)$ be the desired output. At each decoding step, the context consists of the serialized prompt P followed by the tokens generated so far. For the inductive correctness argument, let y be a possibly empty prefix of Y, and write

$$x = P \,\|\, y = (x_1, \ldots, x_L), \tag{25}$$

where $\|$ denotes sequence concatenation.

Let `NULL` be a distinguished symbol that does not occur among the source, instruction, template, or delimiter tokens. For right shifts, nonexistent positions to the left of the sequence are padded with `NULL`.

Listing Listing 2 first computes `prev` by shifting the context x one position to the right, using `NULL` as padding. Thus `prev` contains the preceding token at each context position:

$$\mathtt{prev}_\ell = \begin{cases} \mathtt{NULL}, & \ell = 1, \\ x_{\ell-1}, & 2 \le \ell \le L. \end{cases} \tag{26}$$

The program then applies the fixed tokenwise function `value_at` to each pair consisting of the current token and its predecessor:

$$a = \mathsf{seq_map}(x, \mathtt{prev}, \mathsf{value_at}), \tag{27}$$

$$a_\ell = \mathsf{value_at}(x_\ell, \mathtt{prev}_\ell) \quad (1 \le \ell \le L). \tag{28}$$

The resulting sequence $a = (a_1, \ldots, a_L)$ is `values` in Listing Listing 2. Each a_ℓ stores the value returned when the final lookup selects position ℓ. Expanding the definition of `value_at` gives:

$$a_\ell = \begin{cases} x_\ell, & x_\ell \in V \cup \{\langle \mathsf{ins} \rangle\}, \\ \langle \mathsf{eos} \rangle, & x_\ell = \langle \mathsf{out} \rangle, \\ t_{x_{\ell-1},x_\ell}, & x_{\ell-1} \in K,\ x_\ell \in S, \\ \mathtt{NULL}, & \text{otherwise.} \end{cases} \tag{29}$$

This is precisely `seq_map(x, prev, value_at)`, where `prev` is obtained by shifting x right by one position. For subsequent right shifts of a, we likewise use $a_r = \mathtt{NULL}$ for $r \le 0$.

The sequence a determines what each position returns when selected. We next construct the lookup-key sequence b, corresponding to `keys` in Listing Listing 2. Each position ℓ then represents a lookup entry $b_\ell \mapsto a_\ell$. Thus, to construct the desired relation $\mathcal{L}$, we assign to each stored value the key that should retrieve it.

The program first computes `back1` and `back2` by shifting a right by one and two positions, respectively. With `NULL` padding, their entries satisfy

$$\begin{aligned} \mathtt{back1}_\ell &= a_{\ell-1}, \\ \mathtt{back2}_\ell &= a_{\ell-2}. \end{aligned} \tag{30}$$

Together with the preceding-token sequence prev, these determine the lookup keys:

$$b_\ell = \begin{cases} \texttt{prev}_\ell, & x_\ell \in V,\ \texttt{prev}_\ell \in K, \\ \texttt{back2}_\ell, & x_\ell \in S, \\ \texttt{back1}_\ell, & x_\ell = \langle \mathsf{out} \rangle, \\ \texttt{NULL}, & \text{otherwise.} \end{cases} \tag{31}$$

The first three branches correspond to source, plan, and end in the listing. Their conditions are mutually exclusive, so the two applications of merge combine them into b.

At a source-value position, the current token is v_i and its predecessor is k_i. Since $a_\ell = v_i$, setting $b_\ell = \texttt{prev}_\ell = k_i$ creates the entry $k_i \mapsto v_i$.

After emitting the value for a request, the program must determine which request comes next. We use the preceding output template to identify the completed request: distinct requested pairs have distinct template tokens. Accordingly, the lookup entry $T_{j-1} \mapsto T_j$ specifies the next template to emit after the value associated with T_{j-1}. These entries let the program follow the instruction order using the preceding template as its progress marker.

At the style position of request j, the auxiliary value is $a_\ell = T_j$. To retrieve this template after completing the preceding request, its lookup key must be T_{j-1}. For $j = 1$, the token two positions earlier is $\langle \mathsf{ins} \rangle$, whose auxiliary value is T_0. For $j > 1$, the position two places earlier contains the preceding instruction's style s_{j-1}, whose auxiliary value is T_{j-1}. Hence, in both cases,

$$b_\ell = a_{\ell-2} = T_{j-1}, \qquad a_\ell = T_j, \tag{32}$$

which creates the entry $T_{j-1} \mapsto T_j$.

At the output delimiter, the auxiliary value is $a_\ell = T_{m+1} = \langle \mathsf{eos} \rangle$. When $m > 0$, the immediately preceding position contains the final instruction's style, whose auxiliary value is T_m. Thus $b_\ell = a_{\ell-1} = T_m$ creates the termination entry $T_m \mapsto T_{m+1}$. When $m = 0$, the preceding token is $\langle \mathsf{ins} \rangle$, so the same rule creates $T_0 \mapsto T_1$, namely $\langle \mathsf{ins} \rangle \mapsto \langle \mathsf{eos} \rangle$.

All other lookup keys are NULL. In particular, generated templates are neither styles nor delimiters, and generated values are preceded by templates rather than source keys. They therefore introduce no additional lookup entries. The pairs (b_ℓ, a_ℓ) with non-null keys are exactly the entries of $\mathcal{L}$.

Let $\mathcal{T}$ denote the template vocabulary. The fixed dictionary FIELD in Listing Listing 2 maps each template to the source key whose value it requires:

$$\texttt{FIELD}[t_{q,s}] = q. \tag{33}$$

Thus, after emitting a template, the program can identify the source value to retrieve. Next, the program constructs the query sequence ρ, corresponding to queries in the listing, by applying query_at to the current token and its predecessor:

$$\rho = \mathsf{seq_map}(x, \texttt{prev}, \mathsf{query_at}), \tag{34}$$

$$\rho_\ell = \begin{cases} T_0, & x_\ell = \langle \mathsf{out} \rangle, \\ \texttt{FIELD}[x_\ell], & x_\ell \in \mathcal{T}, \\ \texttt{prev}_\ell, & x_\ell \in V,\ \texttt{prev}_\ell \in \mathcal{T}, \\ \texttt{NOQUERY}, & \text{otherwise.} \end{cases} \tag{35}$$

At $\langle \mathsf{out} \rangle$, no output has yet been generated. The query is therefore $T_0 = \langle \mathsf{ins} \rangle$, which retrieves the first template through the entry $T_0 \mapsto T_1$. And at a generated template $T_j = t_{q_j,s_j}$, FIELD returns q_j. The query therefore retrieves the required source value through the entry $q_j \mapsto v_{q_j}$. Lastly, at a generated value v_{q_j}, the preceding token is T_j. The query is therefore T_j, which retrieves T_{j+1} through the instruction-transition entry $T_j \mapsto T_{j+1}$. This advances to the next requested template, or to the end symbol when $j = m$.

All other positions receive NOQUERY, a reserved symbol distinct from NULL and every other lookup key. Consequently, these queries select no entries, including positions with null lookup keys.

Finally, the program evaluates

$$z = \mathsf{kqv}(b, \rho, a, \mathsf{eq}, \text{default} = \texttt{NULL}), \tag{36}$$

where eq is the equality predicate $\mathsf{eq}(u, v) = (u = v)$. This is the final `kqv` call in the listing, with `keys`, `queries`, and `values` represented by b, ρ, and a, respectively.

At position ℓ, this call selects the positions

$$\mathcal{M}_\ell = \{r : 1 \le r \le \ell,\ b_r = \rho_\ell\}. \tag{37}$$

It returns the mean of their auxiliary values, converted to an integer, or `NULL` if the selection is empty. The autoregressive decoder uses the final component z_L as the next-token prediction and appends it to the context.

For every required next-token query, the selection contains exactly one position. Indeed, the source keys $k_1, \ldots, k_n$ are unique. The transition keys $T_0, \ldots, T_m$ are also distinct, because the requested templates are distinct and T_0 is a reserved delimiter. Moreover, source keys, templates, and delimiters have disjoint vocabularies, and generated positions have null lookup keys. Thus no additional position matches a required query, and averaging returns its unique lookup value exactly.

All these entries lie in the prompt P. Since output generation starts at its final delimiter, every required entry is available under the causal restriction $r \le \ell$. This restriction includes the current position, so an entry stored at $\langle \mathsf{out} \rangle$ is already available when generation begins.

The resulting next-token behavior is therefore

Context ending	ρ_L	z_L
$\langle \mathsf{out} \rangle$	T_0	T_1
T_j	q_j	v_{q_j}
(T_j, v_{q_j})	T_j	T_{j+1}

(38)

The last two rows apply for $1 \le j \le m$. Starting from the empty generated prefix, the first row produces the first template. The remaining rows then alternate between retrieving a source value and advancing to the next template. By induction on the generated prefix, for $m \ge 1$ the decoder produces

$$T_1, v_{q_1}, \ldots, T_m, v_{q_m}, T_{m+1}, \tag{39}$$

which is $\mathsf{rewrite}(D, I)$ followed by $\langle \mathsf{eos} \rangle$. When $m = 0$, the first row immediately returns $T_1 = \langle \mathsf{eos} \rangle$. Repeated source values cause no ambiguity: after emitting a value, the program identifies the completed request by its preceding template.

□

C.2 Proof of Proposition 5.1

Proof. We assume $|Z| = |A| = 2^m$ and $|B| = 2^{4m}$, and identify elements of each set with their binary indices. For $z \in Z$ and $a \in A$, write

$$\mathbf{x}_{z,a} \coloneqq (k_{\mathsf{seed}}, z, k_{\mathsf{in}}, a, k_{\mathsf{out}}, f_z(a)), \qquad \mathbf{y}_{z,a} \coloneqq \mathsf{rewrite}(\mathbf{x}_{z,a}, I) = (t_{\mathsf{in},s}, a, t_{\mathsf{out},s}, f_z(a)),$$

and let $\mathbf{x}_z, \mathbf{y}_z$ denote the corresponding random documents with $a \sim \mathsf{U}(A)$. For $p \in P$, let $L(p; \mathbf{x}_z \mid m) \coloneqq |p| + \mathbb{E}_x[-\log p(\mathbf{x}_z \mid m)]$, so that $p^\star(\mathbf{x}_z)$ minimizes $L(\,\cdot\,; \mathbf{x}_z \mid m)$. We establish two bounds over a uniformly chosen seed $z \sim \mathsf{U}(Z)$ and then select a seed satisfying both.

Claim 1: at least $3/4$ of seeds induce bounded epiplexity.

Let p_{base} be a baseline model that samples $z' \sim \mathsf{U}(Z)$ and $a' \sim \mathsf{U}(A)$ independently and outputs $\mathbf{x}_{z',a'}$. Since the document contains its seed, we have $p_{base}(\mathbf{x}_{z,a} \mid m) = 2^{-2m}$. And since p_{base} does not depend on the seed, its program length is a constant $|p_{base}| = c$, so $\min_{p \in P} L(p; \mathbf{x}_z \mid m) \le 2m + c$.

For any admissible p, define $s_p(z) = \sum_{x \in X} p(\mathbf{x}_{z,a} \mid m)$. The sets $\{\mathbf{x}_{z,a} : a \in A\}$ are disjoint for different seeds, so $\sum_{z \in Z} s_p(z) \le 1$. By Jensen's inequality,

$$\mathbb{E}_x[-\log p(\mathbf{x}_z \mid m)] \ge -\log\big(2^{-m} s_p(z)\big) = m - \log s_p(z). \tag{40}$$

If p beats or ties p_{base}, then $|p| + m - \log s_p(z) \le 2m + c$, i.e., $s_p(z) \ge 2^{|p| - m - c}$. Since $\sum_z s_p(z) \le 1$, at most $2^{m+c-|p|}$ seeds satisfy this, and since $|Z| = 2^m$,

$$\Pr_z[L(p; \mathbf{x}_z \mid m) \le 2m + c] \le 2^{c - |p|}. \tag{41}$$

Taking a union bound over all programs of length at least C, we have

$$\Pr_z[\exists p : |p| \geq C,\ L(p; \mathbf{x}_z \mid m) \leq 2m + c] \leq 2^c \sum_{|p| \geq C} 2^{-|p|}. \tag{42}$$

Since the program code is globally prefix-free, Kraft's inequality implies that the total weight of programs of length at least C tends to zero as C grows. Choose a constant C, independent of m, such that the right-hand side is less than $1/4$. Since every MDL-optimal model satisfies $L(p^\star; \mathbf{x}_z \mid m) \leq L(p_{base}; \mathbf{x}_z \mid m) \leq 2m + c$, for at least $3/4$ of all seeds every MDL-optimal source model satisfies $|p^\star(\mathbf{x}_z)| < C$.

Claim 2: a $(1 - o(1))$ fraction of seeds have high rewritten epiplexity.

A model that hard-codes z, samples $a \sim \mathsf{U}(A)$, and outputs $\mathbf{y}_{z,x}$ has program length $m + O(\log m)$ and expected log loss $\log |A| = m$. Hence, for a suitable bound D_m independent of z,

$$\min_{p \in P} L(p; \mathbf{y}_z \mid m) \leq D_m, \qquad D_m = 2m + O(\log m). \tag{43}$$

Applying the PRF bound of Finzi et al. (2026, Appendix A.4, Eq. (32)) to the pair $(a, f_z(a)) \in A \times B$ (the template symbols in $\mathbf{y}_z$ are fixed), with $\log |A \times B| = 5m$ and entropy m, gives for every admissible p

$$\Pr_z[\mathbb{E}_x[-\log p(\mathbf{y}_z \mid m)] \leq D_m] \leq 2\varepsilon(m) + 2^{1+2D_m - 5m}. \tag{44}$$

Here $\varepsilon(m)$ is negligible and can be chosen uniformly over admissible programs of length at most $\lfloor \log m \rfloor$ by nonuniform security. Since every MDL minimizer has expected log loss at most D_m, a union bound over these short programs gives

$$\Pr_z[|p^\star(\mathbf{y}_z)| \leq \lfloor \log m \rfloor] \leq 2^{\lfloor \log m \rfloor + 1} \left(2\varepsilon(m) + 2^{1+2D_m - 5m}\right) = o(1). \tag{45}$$

Using the above two claims, the same seed can satisfy both bounds. Indeed,

$$\Pr_z[|p^\star(\mathbf{x}_z)| < C,\ |p^\star(\mathbf{y}_z)| > \lfloor \log m \rfloor] \geq \frac{3}{4} - o(1) > 0 \tag{46}$$

for sufficiently large m. Choose such an m with $\lfloor \log m \rfloor > C$ and fix a seed z in this intersection. For this seed, every MDL-optimal source model has length below C, whereas every MDL-optimal rewritten model has length above $\lfloor \log m \rfloor$. Therefore the fixed deterministic rewrite strictly increases epiplexity, with the stated gap. □

C.3 Proof of Proposition 5.2

Proof. We use the notation and the cardinality assumption of the proof of Proposition 5.1, now with $\mathbf{y}_{z,a} \coloneqq \mathsf{rewrite}(\mathbf{x}_{z,a}, I) = (t_{\mathsf{in},s}, a, t_{\mathsf{out},s}, f_z(a), t_{\mathsf{seed},s}, z)$ given by (13), and with $p^\star$ taken over the causal model class P.

Causal models can predict the seed and input uniformly, then compute the output from their observed values. Hence, the same Kraft inequality in the proof of Proposition 5.1 also holds for the causal class. Therefore, we get $\Pr_z[|p^\star(\mathbf{x}_z)| < C] \geq 3/4$ for a sufficiently large constant C.

For a causal model p on the rewritten document, let $q_{\mathrm{pre}}(a, b \mid m)$, $(a, b) \in A \times B$, denote its normalized distribution on the input–output prefix, suppressing fixed template symbols. This distribution is efficiently evaluable without the final seed, and for $\mathbf{y} = (t_{\mathsf{in},s}, a, t_{\mathsf{out},s}, b, t_{\mathsf{seed},s}, z')$,

$$p(\mathbf{y} \mid m) = q_{\mathrm{pre}}(a, b \mid m)\, p(z' \mid a, b, m) \leq q_{\mathrm{pre}}(a, b \mid m). \tag{47}$$

A causal model storing z represents $\mathbf{y}_z$ exactly, so its optimal MDL score is at most $D_m = 2m + O(\log m)$. By (47), every optimal model consequently has expected prefix log loss $\mathbb{E}_x[-\log q_{\mathrm{pre}}(a, f_z(a) \mid m)] \leq D_m$.

Applying the same union bound argument used to derive (45) to causal models of length at most $\lfloor \log m \rfloor$, we obtain

$$\Pr_z[|p^\star(\mathbf{y}_z)| \leq \lfloor \log m \rfloor] \leq 2^{\lfloor \log m \rfloor + 1} \left(2\varepsilon(m) + 2^{1+2D_m - 5m}\right) = o(1), \tag{48}$$

where $\varepsilon(m)$ is negligible by nonuniform PRF security. The two good-seed sets therefore intersect with probability at least $3/4 - o(1) > 0$. For every sufficiently large m with $\lfloor \log m \rfloor > C$, choosing a seed z in this intersection proves the claim. □

C.4 Proof of Proposition 5.3

Proof. Throughout, abbreviate $\bar{\mathbf{x}} \coloneqq \mathsf{rewrite}(\mathbf{x}, I_{\mathsf{fact}})$ and $\widetilde{\mathbf{x}} \coloneqq \mathsf{rewrite}(\mathbf{x}, I_{\mathsf{fic}})$. All probabilities over seeds use independent $z \sim \mathsf{U}(Z)$ and $w \sim \mathsf{U}(W)$; each data distribution has one fixed pair (z, w), and the random choice of seeds serves only to show that a suitable pair exists.

Claim 1: $\mathbf{x}$ and $\bar{\mathbf{x}}$ have bounded epiplexity at least $3/4$ of seeds.

Let p_{base} be the constant-length model that samples $z' \sim \mathsf{U}(Z)$ and $a \sim \mathsf{U}(A)$ independently, and outputs $\mathbf{x}' = (k_{\mathsf{seed}}, z', k_{\mathsf{in}}, a, k_{\mathsf{out}}, f_{z'}(a))$. And let $\bar{p}_{base}$ be the constant-length model that samples the same (z', a) and outputs $\mathsf{rewrite}(\mathbf{x}', I_{\mathsf{fact}})$. Both lie in P, since f is polynomial-time computable and I_{fact} only copies values and prints fixed templates, and

$$p_{base}(\mathbf{x} \mid m) = \bar{p}_{base}(\bar{\mathbf{x}} \mid m) = \frac{1}{|Z||A|}, \quad \forall z \in Z. \tag{49}$$

Hence if $p, \bar{p} \in P$ beats or ties p_{base} and $\bar{p}_{base}$, then $L(p; \mathbf{x}|m) \le \log|Z| + m + |p_b|$ and $L(\bar{p}; \bar{\mathbf{x}}|m) \le \log|Z| + m + |\bar{p}_b|$, which construct the first bound in (42). Set $c \coloneqq \max\{|p_b|, |\bar{p}_b|\}$.

Let $\mathbf{v}$ denote either $\mathbf{x}$ or $\bar{\mathbf{x}}$. Since z is printed in $\mathbf{v}$, the supports of $\mathbf{v}$ are disjoint across seeds. For any $p \in P$ let

$$s_p(z) \coloneqq \sum_{x \in X} p(\mathbf{v} \mid m), \quad \text{so that} \quad \sum_{z \in Z} s_p(z) \le 1. \tag{50}$$

By Jensen's inequality, $\mathbb{E}_x[-\log p(\mathbf{v} \mid m)] \ge m - \log s_p(z)$. For any $p \in P$, if p beats or ties the baseline of length at most c, then $|p| + m - \log s_p(z) \le \log|Z| + m + c$, that is,

$$s_p(z) \ge \frac{2^{|p|-c}}{|Z|}, \tag{51}$$

and since the $s_p(z)$ sum to at most 1, at most a $2^{c-|p|}$ fraction of seeds can satisfy this. By the prefix-free assumption, $\sum_p 2^{-|p|} \le 1$, so we can choose a constant C, independent of m, with

$$2^c \sum_{p:\, |p| \ge C} 2^{-|p|} < \frac{1}{4}. \tag{52}$$

A union bound over all programs of length at least C shows that, for every choice of minimizers,

$$\Pr_z\big[|p^\star(\mathbf{x})| < C\big] \ge \frac{3}{4}, \quad \Pr_z\big[|p^\star(\bar{\mathbf{x}})| < C\big] \ge \frac{3}{4}. \tag{53}$$

Claim 2: a (1-$o(1)$) fraction of seeds have high epiplexity on $\widetilde{\mathbf{x}}$.

Let p_w be the model that stores w, samples $z' \sim \mathsf{U}(Z)$ and $a \sim \mathsf{U}(A)$, and outputs $\mathsf{rewrite}(\mathbf{x}', I_{\mathsf{fic}})$ computed with seeds (z', w). It lies in P because f and π are polynomial-time computable. Its length is $\log|W| + O(\log m)$, and $p_w(\widetilde{\mathbf{x}} \mid m) = 1/(|Z||A|)$ for every z, so its expected log loss is $\log|Z| + m$. Fix a computable integer, uniform in (z, w),

$$D_m \coloneqq \log|Z| + m + \log|W| + O(\log m) \quad \text{such that} \quad L^\star(\widetilde{\mathbf{x}}) \le D_m, \tag{54}$$

which proves the second bound in (45). In particular, every minimizer $p^\star(\widetilde{\mathbf{x}})$ satisfies $\mathbb{E}[-\log p^\star(\widetilde{\mathbf{x}} \mid m)] \le D_m$.

Claim 3: short models cannot achieve this bound for most seed pairs.

Define the polynomial-time template

$$\rho(z, a, a') \coloneqq (t_{\mathsf{seed},s}, z, t_{\mathsf{in},s}, a, t_{\mathsf{out},s}, f_z(a), t_{\mathsf{in,fic}}, a', t_{\mathsf{out,fic}}, f_z(a')), \tag{55}$$

which is injective in (z, a, a') and satisfies $\widetilde{\mathbf{x}} = \rho(z, a, \pi_w(a))$. Let $\mathcal{P}_m$ be the set of programs $p \in P$ with $|p| \le \lfloor \log m \rfloor$. It has at most $2^{\lfloor \log m \rfloor + 1} = O(m)$ elements and can be enumerated in polynomial time. For $p \in \mathcal{P}_m$ define

$$H_p \coloneqq \big\{(z, a, a') \in Z \times A \times A :\ p(\rho(z, a, a') \mid m) \ge 2^{-(D_m + \lfloor m/3 \rfloor)}\big\}. \tag{56}$$

Because ρ is injective and $\sum_{\mathbf{v}} p(\mathbf{v} \mid m) \le 1$, we have $|H_p| \le 2^{D_m + \lfloor m/3 \rfloor}$ for every $p \in \mathcal{P}_m$.

To use the assumption that π is not distinguishable, we combine the short models into a single procedure that tries to distinguish π_w from the random permutation.

Let $\mathrm{Perm}(A)$ be the set of permutations of A and $\Pi \sim \mathsf{U}(\mathrm{Perm}(A))$ a uniformly random permutation. For a map $g : A \to A$, define the test T_g as follows: sample $z \sim \mathsf{U}(Z)$ and $a \sim \mathsf{U}(A)$. Then compute $a' = g(a)$, and output 1 if $(z, a, a') \in H_p$ for at least one $p \in \mathcal{P}_m$, and 0 otherwise. When $g = \pi_w$, the triple $(z, a, \pi_w(a))$ corresponds to the fictional document $\rho(z, a, \pi_w(a))$, so T_{π_w} checks whether some short model assigns high probability to it. The test evaluates g once and omputes $O(m)$ likelihoods, each within the polynomial budget of P, so it runs in polynomial time. Its steps are fixed by m alone: it does not know w, and it learns about w only through the value $g(a)$. The indistinguishability assumption therefore applies to T and gives a negligible $\varepsilon(m)$ such that

$$\Pr_w\big[T_{\pi_w} = 1\big] \le \Pr_\Pi\big[T_\Pi = 1\big] + \varepsilon(m). \tag{57}$$

We now bound the two sides of (57). If $g = \Pi$, then $\Pi(a)$ is uniform on A for each fixed a, so $(z, a, \Pi(a))$ is uniform on $Z \times A \times A$. A union bound over $\mathcal{P}_m$, together with $D_m = \log|Z| + m + \log|W| + O(\log m)$ and $|W| \le 2^{m/3}$, yields

$$\Pr_\Pi\big[T_\Pi = 1\big] \le \sum_{p \in \mathcal{P}_m} \frac{|H_p|}{|Z|\,|A|^2} \le O(m) \cdot 2^{\log|W| + \lfloor m/3 \rfloor + O(\log m) - m} \le 2^{-m/3 + O(\log m)}. \tag{58}$$

If $g = \pi_w$, let G be the set of pairs (z, w) for which some $p \in \mathcal{P}_m$ satisfies $\mathbb{E}_a[-\log p(\widetilde{\mathbf{x}} \mid m)] \le D_m$, and fix such a pair and such a p. Markov's inequality applied to the nonnegative variable $-\log p(\widetilde{\mathbf{x}} \mid m)$ gives

$$\Pr_a\big[(z, a, \pi_w(a)) \in H_p\big] = \Pr_a\big[-\log p(\widetilde{\mathbf{x}} \mid m) \le D_m + \lfloor m/3 \rfloor\big] \tag{59}$$

$$\ge 1 - \frac{D_m}{D_m + \lfloor m/3 \rfloor} = \frac{\lfloor m/3 \rfloor}{D_m + \lfloor m/3 \rfloor} \ge \frac{1}{9} \tag{60}$$

for all sufficiently large m, because $D_m \le 2m + m/3 + O(\log m)$. The test outputs 1 whenever $(z, a, \pi_w(a)) \in H_p$ for this particular p, so averaging over the z and a sampled inside the test and over w gives

$$\Pr_w\big[T_{\pi_w} = 1\big] \ge \frac{1}{9} \Pr_{z,w}[G]. \tag{61}$$

Combining with (57) and (58),

$$\Pr_{z,w}[G] \le 9\big(\varepsilon(m) + 2^{-m/3 + O(\log m)}\big) = o(1). \tag{62}$$

By Claim 2, every MDL-optimal model of $\widetilde{\mathbf{x}}$ has expected log loss at most D_m, so $|p^\star(\widetilde{\mathbf{x}})| \le \lfloor \log m \rfloor$ implies $(z, w) \in G$. Therefore

$$\Pr_{z,w}\big[|p^\star(\widetilde{\mathbf{x}})| \le \lfloor \log m \rfloor\big] \le \Pr_{z,w}[G] = o(1). \tag{63}$$

Combining the three claims, by (53) and (63),

$$\Pr_{z,w}\big[|p^\star(\mathbf{x})| < C, |p^\star(\bar{\mathbf{x}})| < C, |p^\star(\widetilde{\mathbf{x}})| > \lfloor \log m \rfloor\big] \ge 1 - \frac{1}{4} - \frac{1}{4} - o(1) > 0, \tag{64}$$

and $C < \lfloor \log m \rfloor$ for all sufficiently large m. Any pair (z, w) in this event satisfies (17). □

D EXISTING LLM GENERATION PROMPTS

In Section 4, we focus our analysis to rewriting operations on pretraining data where relatively smaller rewriting models empirically perform competitively to larger ones in terms of improving downstream task performance, which constitute some of the most counterintuitive observations on rewriting (Section 1). Across the literature (Maini et al., 2024; Nguyen et al., 2025; Su et al., 2025; Kang et al., 2025; Maini et al., 2025; Niklaus et al., 2026), we identify the following such cases: Wikipedia-style rewriting (Maini et al., 2024; Kang et al., 2025), QA rewriting (Kang et al., 2025; Niklaus et al., 2026), and summarization (Niklaus et al., 2026), shown in Table 1.

- Maini et al. (2024) have shown that Wikipedia-style rewriting using Qwen 1.8B and Mistral 7B achieves lower downstream perplexity on the Pile corpus than rewriting using Vicuna 13B.
- Kang et al. (2025) have shown that Wikipedia-style and QA rewriting using Llama 3 8B achieve lower and similar downstream perplexity on the Pile, respectively, compared to Llama 3 70B.
- Niklaus et al. (2026) have shown that summarization and QA rewriting using Gemma 3 1B achieve better downstream task performance across 12 benchmarks than rewriting using Gemma 3 12B.

We list the full rewriting prompts used by the prior work cited in Section 4, quoted from the original papers. Each asks a language model to restate the input in a prescribed form while preserving its content, which is the setting our analysis models as simple rewriting.

D.1 Prompts from Maini et al. (2024)

Wikipedia style

```
A chat between a curious user and an artificial intelligence assistant. The assistant
gives helpful, detailed, and polite answers to the questions. [USER]: For the following
paragraph give me a diverse paraphrase of the same in high quality English language as in
sentences on Wikipedia: [TEXT]
```

D.2 Prompts from Kang et al. (2025)

Wikipedia style

```
System Prompt: Provide direct and detailed response to the instructions without adding
additional notes.
[USER]: For the following document, regardless of its original content or formatting,
write a full article of the same content in high quality English language as in texts on
Wikipedia: [TEXT]. Provide the rephrased article without any additional notes. Long
article with full length and complete details. Rephrased article:
```

QA style

```
System Prompt: Provide direct and detailed response to the instructions without adding
additional notes.
[USER]: For the following document, regardless of its original content or formatting,
convert it into a comprehensive list of question-answer pairs with multiple tags of "
Question:" followed by "Answer:", where questions and answers cover complete information
of the original document. Document: [TEXT]. Provide the converted question-answer pairs
without any additional notes. Question-answer pairs with corresponding tags ("Question:",
"Answer:"):
```

D.3 Prompts from Niklaus et al. (2026)

QA style

```
Rewrite the document as a comprehensive FAQ (Frequently Asked Questions). Extract or infer
 the key questions a reader would have about this topic, then provide clear, direct
answers. Order questions logically-from foundational to advanced, or by topic area. Each
answer should be self-contained and understandable without reference to other answers.
Ensure the FAQ works as a standalone document. Output only the FAQ, nothing else.
Document:
[TEXT]
```

Summarization style

```
Summarize the following text. Write a standalone summary without referencing the text.
Directly start with the summary. Do not say anything else.
Document:
[TEXT]
```

E REWRITING PROMPTS

We list the prompts with which TinyLlama-1.1B-Chat generates the rewritten data used in our experiments, together with the sampling settings for each task.

E.1 KNOWLEDGE EXTRACTION

We follow the bioR generation prompt of Allen-Zhu & Li (2023a), limiting the biography to at most 80 words. Rewrites are sampled with temperature 0.5 and top-p 0.9. For each person we draw up to 20 samples and keep the first biography that states all six attributes correctly, as judged by the criterion of Section 6.1. We repeat this four times to obtain four biographies per person.

Biography (Permutation + Rephrasing)

```
Write a 80 words biography about a person whose name is Anya Briar Forger. The person was
born on October 2, 1996. The person graduated from MIT. The person majored in Physics
there. The person worked in Palo Alto for Meta. The person is a native of Princeteon, NJ.
```

E.2 KNOWLEDGE REVERSAL

Rewrites are first decoded greedily. Over 99.5% of outputs match the target statement on the first attempt. The remaining failures come mostly from misspellings of less common entity names, which the model replaces with a more familiar form (e.g., *Charlot* $\rightarrow$ *Charlotte*). For these inputs, we draw up to five samples per input with temperature 0.8 and top-p 0.95 and keep the first that matches. The example names in the prompt do not appear in any input facts.

Head identity (h, r_{id}, h)

```
Rewrite the sentence. Take the first name in the sentence and call it N. Answer exactly: N
 has a self named N. The output contains only N, written twice; the other name must not
appear.
Examples:
Input:
Priya has a husband named Rahul.
Output:
Priya has a self named Priya.
Input:
Sven has a wife named Ingrid.
Output:
Sven has a self named Sven.
Input:
Zofia has a husband named Tariq.
Output:
Zofia has a self named Zofia.
Input:
Kenji has a wife named Yuki.
Output:
Kenji has a self named Kenji.
Input:
Mei Ling has a husband named Wei.
Output:
Mei Ling has a self named Mei Ling.
Input:
Ravi has a wife named Ines.
Output:
Ravi has a self named Ravi.
```

Tail identity (t, r_{id}, t)

```
Rewrite the sentence. Take the last name in the sentence and call it N. Answer exactly: N
has a self named N. The output contains only N, written twice; the other name must not
appear.
Examples:
Input:
Priya has a husband named Rahul.
Output:
```

```
Rahul has a self named Rahul.
Input:
Sven has a wife named Ingrid.
Output:
Ingrid has a self named Ingrid.
Input:
Zofia has a husband named Tariq.
Output:
Tariq has a self named Tariq.
Input:
Kenji has a wife named Yuki.
Output:
Yuki has a self named Yuki.
Input:
Mei Ling has a husband named Wei Ling.
Output:
Wei Ling has a self named Wei Ling.
Input:
Ravi has a wife named Ines.
Output:
Ines has a self named Ines.
```

E.3 TWO-HOP COMPOSITION

E.3.1 IDENTITY

Rewrites are first decoded greedily. Over 99% of outputs match the target statement on the first attempt. The remaining failures come mostly from entity names the model tokenizes into rare pieces (e.g., *Warrmont*), which it occasionally misspells. For these inputs, we draw up to two samples per input with temperature 0.8 and top-p 0.95 and keep the first that matches. The example names in the prompt do not appear in any input facts.

Bridge identity (b, r_{id}, b)

```
Rewrite the sentence using this exact rule.
The input format is:
SUBJECT has a RELATION named OBJECT.
The output format is:
SUBJECT has a self named SUBJECT.
Rules:
- Copy SUBJECT exactly and write it twice.
- Replace RELATION with the word "self". RELATION never appears in the output.
- Delete OBJECT.
Examples:
Input:
Marlow has a friend named Deveraux.
Output:
Marlow has a self named Marlow.
Input:
Petersham has a mentor named Ashfield.
Output:
Petersham has a self named Petersham.
Input:
Wren has a lawyer named Coleridge.
Output:
Wren has a self named Wren.
Return only one sentence.
```

First-hop identity $(h, r_1, r_{\mathrm{id}}, b)$

```
Rewrite the sentence using this exact rule.
The input format is:
SUBJECT has a RELATION named OBJECT.
The output format is:
SUBJECT has a RELATION whose self is named OBJECT.
Rules:
- Replace the word "named" with "whose self is named". Change nothing else.
- Keep SUBJECT, RELATION and OBJECT unchanged.
Examples:
Input:
Marlow has a father named Deveraux.
```

```
Output:
Marlow has a father whose self is named Deveraux.
Input:
Petersham has a mother named Ashfield.
Output:
Petersham has a mother whose self is named Ashfield.
Input:
Wren has a brother named Coleridge.
Output:
Wren has a brother whose self is named Coleridge.
Return only one sentence.
```

Second-hop identity $(b, r_{\mathrm{id}}, r_2, t)$

```
Rewrite the sentence using this exact rule.
The input format is:
SUBJECT has a RELATION named OBJECT.
The output format is:
SUBJECT has a self whose RELATION is named OBJECT.
Rules:
- Insert "self whose" after "has a". Change nothing else.
- Keep SUBJECT, RELATION and OBJECT unchanged.
Examples:
Input:
Marlow has a friend named Deveraux.
Output:
Marlow has a self whose friend is named Deveraux.
Input:
Petersham has a mentor named Ashfield.
Output:
Petersham has a self whose mentor is named Ashfield.
Input:
Wren has a lawyer named Coleridge.
Output:
Wren has a self whose lawyer is named Coleridge.
Return only one sentence.
```

E.3.2 BRIDGE PERMUTATION

The permutation π is fixed in advance and supplied to the rewriting model as part of the input. The model does not choose it. Rewrites are first decoded greedily. Over 99% of outputs match the target statement on the first attempt. The remaining failures come mostly from entity names the model tokenizes into rare pieces (e.g., *Warrmont*), which it occasionally misspells. For these inputs, we draw up to eight samples per input with temperature 0.8 and top-p 0.95 and keep the first that matches.

Bridge atomic $(b, r_\pi, \pi(b)), (\pi(b), r_\pi^{-1}, b)$

```
Rewrite the sentence using this exact rule.
The input format is:
SUBJECT pair OBJECT.
The output format is:
SUBJECT has a giver named OBJECT.
Rules:
- Keep SUBJECT at the front and OBJECT at the end. Never swap them.
- Copy both names letter for letter. Never reuse one name twice.
Examples:
Input:
Marlow pair Deveraux.
Output:
Marlow has a giver named Deveraux.
Input:
Petersham pair Ashfield.
Output:
Petersham has a giver named Ashfield.
Input:
Wren pair Coleridge.
Output:
Wren has a giver named Coleridge.
Return only one sentence.
```

Bridge round-trip $(b, r_\pi, r_\pi^{-1}, b), (\pi(b), r_\pi^{-1}, r_\pi, \pi(b))$

```
Rewrite the sentence using this exact rule.
The input format is:
SUBJECT has a giver named OBJECT.
The output format is:
SUBJECT has a giver whose receiver is named SUBJECT.
Rules:
- Keep SUBJECT unchanged.
- Use SUBJECT twice, not OBJECT.
Examples:
Input:
Marlow has a giver named Deveraux.
Output:
Marlow has a giver whose receiver is named Marlow.
Input:
Petersham has a giver named Ashfield.
Output:
Petersham has a giver whose receiver is named Petersham.
Input:
Wren has a giver named Coleridge.
Output:
Wren has a giver whose receiver is named Wren.
Return only one sentence.
```

Bridge first-hop $(h, r_1, r_\pi, \pi(b))$

```
Merge the two input lines into one sentence using this exact rule.
The input format is:
SUBJECT has a RELATION named BRIDGE.
BRIDGE has a giver named TARGET.
The output format is:
SUBJECT has a RELATION whose giver is named TARGET.
Rules:
- SUBJECT is the first name on line 1. It stays at the FRONT.
- RELATION is the word on line 1 between "has a" and "named". Copy it
  unchanged.
- TARGET is the last name on line 2, even a short one. It goes at the END.
- BRIDGE is the shared name that ends line 1 and begins line 2. It must
  NOT appear in the output at all.
- Copy every name letter for letter.
Examples:
Input:
Marlow has a father named Deveraux.
Deveraux has a giver named Petersham.
Output:
Marlow has a father whose giver is named Petersham.
Input:
Ashfield has a spouse named Wren.
Wren has a giver named Coleridge.
Output:
Ashfield has a spouse whose giver is named Coleridge.
Input:
Ashbury has a sister named Tilden.
Tilden has a giver named Kesteven.
Output:
Ashbury has a sister whose giver is named Kesteven.
Return only one sentence.
```

Bridge second-hop $(\pi(b), r_\pi^{-1}, r_2, t)$

```
Merge the two input lines into one sentence using this exact rule.
The input format is:
SUBJECT has a receiver named BRIDGE.
BRIDGE has a RELATION named TARGET.
The output format is:
SUBJECT has a receiver whose RELATION is named TARGET.
Rules:
- SUBJECT is the first name on line 1. It stays at the FRONT.
- RELATION is the word on line 2 between "has a" and "named". Copy it
  unchanged.
- TARGET is the last name on line 2, even a short one. It goes at the END.
- BRIDGE is the shared name that ends line 1 and begins line 2. It must
```

```
  NOT appear in the output at all.
- Copy every name letter for letter.
Examples:
Input:
Marlow has a receiver named Deveraux.
Deveraux has a friend named Petersham.
Output:
Marlow has a receiver whose friend is named Petersham.
Input:
Ashfield has a receiver named Wren.
Wren has a doctor named Coleridge.
Output:
Ashfield has a receiver whose doctor is named Coleridge.
Input:
Ashbury has a receiver named Tilden.
Tilden has a mentor named Kesteven.
Output:
Ashbury has a receiver whose mentor is named Kesteven.
Return only one sentence.
```

F DATA EXAMPLES

We give one example of each data type for the three tasks. See Section 6 for how the pretraining and downstream sets are composed from these types.

F.1 KNOWLEDGE EXTRACTION

Pretraining data.

Type	Example
Original	Olivia Mckenzie Johns arrived on December 3, 1834. He feels a deep connection to Baton Rouge, LA. He was mentored by distinguished professors at University of Texas Health Science Center at Houston. He pursued interdisciplinary studies related to Information Technology. He was part of the team at Starbucks. He was part of the workforce driving innovation in Seattle, WA.
Rewrite (template)	Olivia Mckenzie Johns gained insights into the industry while working at Starbucks. He applied their knowledge of Information Technology to real-world problems. He acquired industry knowledge while working in Seattle, WA. He had the opportunity to contribute to the research and innovation ecosystem at University of Texas Health Science Center at Houston. He owes their roots to Baton Rouge, LA. He's birth is celebrated annually on December 3, 1834.
Rewrite (TinyLlama)	Olivia Mckenzie Johns was born on December 3, 1834, in Baton Rouge, Louisiana, to parents William Mckenzie and Mary Ann McKenzie. He was raised in a family of farmers and attended local schools. At the age of 16, Olivia moved to Seattle, Washington, where he worked as a farm laborer. He later joined the Starbucks team in Seattle and worked as a barista. In 1996, he completed his studies at the University of Texas Health Science Center at Houston, where he studied Information Technology.

Downstream query.

Type	Question	Answer
Attribute QA	What is the birth date of Olivia Mckenzie Johns?	December 3, 1834

The same format is used for the remaining five attributes.

F.2 KNOWLEDGE REVERSAL

Pretraining data.

Type	Example
Forward atomic	Sinclair has a husband named Paul.
Reverse atomic	Paul has a wife named Sinclair.
Identity	Paul has a self named Paul.

Downstream query.

Type	Question	Answer
Forward QA	Who is the husband of Sinclair?	Paul
Reverse QA	Who is the wife of Paul?	Sinclair

F.3 TWO-HOP COMPOSITION

Relations. First-hop relations $r_1 \in \{\text{father, mother, brother, sister, spouse}\}$; second-hop relations $r_2 \in \{\text{friend, mentor, boss, doctor, lawyer}\}$; identity relation $r_{id} = \text{self}$; bridge permutation relations $r_\pi = \text{giver}$, $r_\pi^{-1} = \text{receiver}$.

Pretraining data.

Type	Example
Atomic (r_1)	Mereden has a spouse named Anthony.
Atomic (r_2)	Yates has a lawyer named Khatun.
Two-hop	Garbeck has a spouse whose boss is named Pemwick.
Identity	Caldman has a self named Caldman.
Identity two-hop (r_1, self)	Warrford has a mother whose self is named Jadyn.
Identity two-hop (self, r_2)	Rockwood has a self whose lawyer is named Ingman.
Bridge atomic	Leech has a giver named Larkbury. Larkbury has a receiver named Leech.
Bridge round-trip	Leech has a giver whose receiver is named Leech. Larkbury has a receiver whose giver is named Larkbury.
Bridge two-hop (r_1, r_π)	Pierdon has a father whose giver is named Wynley.
Bridge two-hop (r_π^{-1}, r_2)	Nick has a receiver whose boss is named Ramholt.

Downstream query.

Type	Question	Answer
Two-hop QA	Who is the boss of the spouse of Garbeck?	Pemwick

The same format is used for every two-hop questions.

G EXPERIMENTAL DETAILS

G.1 BLOCK EPIPLEXITY

To estimate Epiplexity for a multi-epoch dataset at the same cost as pretraining, we fix the pretraining architecture, optimization settings, and total training budget. The entire dataset is split into $K = 16$ blocks $b_1, \ldots, b_K$, with geometrically growing cumulative sizes. In phase i, b_i is first evaluated using θ_{i-1}, after which training continues on $D_i = \bigcup_{j=1}^{i} b_j$ to obtain θ_i. Training steps for phase i are allocated in proportion to $|D_i|$. Epiplexity is then estimated as

$$\widehat{S} = \sum_{i=1}^{K} n_i \left[L(\theta_{i-1}; b_i) - L(\theta_K; \mathcal{D}_{\text{val}}) \right], \tag{65}$$

where θ_0 is randomly initialized, n_i is the token count of b_i, L is the mean per-token loss in bits, and $\mathcal{D}_{\text{val}}$ is a held-out validation split drawn from the same distribution as the corresponding dataset and excluded from training. The first term is the prequential code length. The second term uses the validation loss of the final model θ_K to approximate the per-token code length of the data under that model. Their difference therefore approximates the model cost. To make the comparison

independent of dataset size, we match the token counts of the original and rewritten datasets. For the original set, we exclude OOD atomic facts and repeat the ID atomic facts and their task-specific instances until the token count matches that of the rewritten set. Thus, the two sets differ only in the applied transformation, not in the amount of training data.

G.2 Training details

For all training experiments, one of the TinyLlama-1.1B-Chat, TinyLlama-1.1B-Base, or randomly initialized TinyLlama-1.1B models was used.

G.2.1 Knowledge extraction

Data. For the rule-based condition, we generate the four rewrites per ID biography using fixed sentence-template transformations and sentence permutations, corresponding to the Multi and Permutation settings of Allen-Zhu & Li (2023a). For the LLM-based condition, we provide TinyLlama-1.1B-Chat with the individual's name, gender, and six attributes and prompt it to generate a biography describing the individual. For fine-tuning, we construct a QA example querying each attribute of each individual.

For the QA fine-tuning set, we construct questions that query each attribute of an individual.

Pretraining. For pretraining, we pack the biographies into sequences of 512 tokens. We train the model for 100,000 steps with a batch size of 512 using AdamW. The learning rate follows a cosine decay schedule from a maximum of 10^{-3} to a minimum of 10^{-4}. We use a weight decay of 0.1, $\epsilon = 10^{-6}$, and $(\beta_1, \beta_2) = (0.9, 0.999)$.

QA finetuning. Starting from the pretrained checkpoint, we perform LoRA fine-tuning with rank $r = 16$ and scaling parameter $\alpha = 16$ for the query and value projection modules, and $r = 128$ and $\alpha = 128$ for the token embeddings. We set the LoRA dropout to 0 and apply LoRA to the query projection, value projection, and token embedding modules. Each training example is padded with the EOS token to the maximum sequence length within the batch, and the cross-entropy loss is computed only over the answer tokens. We use AdamW with a learning rate of 10^{-4}, cosine-decayed to 10^{-5}, a weight decay of 0.01, $\epsilon = 10^{-6}$, and $(\beta_1, \beta_2) = (0.9, 0.999)$.

G.2.2 Knowledge reversal

Data. We instantiate the knowledge-reversal setting described in Section 6 using husband-wife relation. Entity names are sampled as single-word names from two disjoint halves of a name pool, with each name appearing in exactly one pair. We convert the forward and reverse facts into corresponding QA examples for fine-tuning.

Pretraining. For pretraining experiments on knowledge reversal task, we pad EOS tokens to each training example to the longest one and apply cross-entropy loss only to the answer span. The model is then trained for 50,000 steps with batch size 512. We use AdamW optimizer with a peak learning rate of 4×10^{-4}, weight decay 0.5, $\epsilon = 10^{-6}$, and $(\beta_1, \beta_2) = (0.9, 0.999)$. We used 500 linear warmup steps and a cosine decay to 4×10^{-5}.

QA finetuning. Starting from the pretrained checkpoint, we do full fine-tuning. Similarly to pretraining, we zero pad training examples and apply cross-entropy loss only to the answer token. We use AdamW optimizer with an initial learning rate of 10^{-5} cosine decayed to 10^{-6}, weight decay 0.01, $\epsilon = 10^{-6}$, and $(\beta_1, \beta_2) = (0.9, 0.999)$.

G.2.3 Two-hop composition

Data. We instantiate the two-hop setting described in Section 6 by sampling distinct single-word names for the head, bridge, and tail entities. We use {`father`, `mother`, `brother`, `sister`, `spouse`} as the first-hop relations and {`friend`, `mentor`, `boss`, `doctor`, `lawyer`} as the second-hop relations. We also use {`self`} as the identity relation and {`giver`, `receiver`} as the bridge-permutation relations. For fine-tuning, each two-hop fact is converted into a QA example that queries the tail entity given the head entity and the two relations.

Pretraining. For pretraining, we pad EOS token to each training example to the longest one and apply cross-entropy loss only to the answer span. The model is then trained for 50,000 steps with batch size 512. We use AdamW optimizer with a peak learning rate of 4×2^{-4}, weight decay 0.3, $\epsilon = 10^{-6}$, and $(\beta_1, \beta_2) = (0.9, 0.95)$. We used 100 linear warmup steps and a cosine decay to 10^{-5}.

QA finetuning. Starting from the pretrained checkpoint, we do full fine-tuning. Similarly to pretraining, we zero pad training examples and apply cross-entropy loss only to the answer token. We use AdamW optimizer with an initial learning rate of 10^{-5} cosine decayed to 10^{-6}, weight decay 0.01, $\epsilon = 10^{-6}$, and $(\beta_1, \beta_2) = (0.9, 0.95)$.

G.3 Cross-layer injection into the two-hop query

Queries and pairs. We use the pretraining query format, "h has a r_1 whose r_2 is named". Each pair contains a source query $q = (h, r_1, r_2)$ with bridge b and answer $t = r_2(b)$, and a recipient query $q' = (h', r_1, r_2)$ with bridge $b' \neq b$ and answer $t' = r_2(b') \neq t$. The relations are identical across the pair. We sample $n = 200$ pairs per split and use the same pairs for all three learners.

The split labels refer only to the source query. For ID–ID sources, both constituent edges, (h, r_1, b) and (b, r_2, t), have appeared in training compositions, but the queried composition is held out. For OOD–OOD sources, both edges have appeared only as atomic facts and have never been composed during training. Recipients are selected from the atomic graph subject to the relation-matching and distinct-answer conditions above. Their edges are not restricted to the source's split, and recipient queries are not required to be held out.

Intervention. We intervene at the last token of r_1. Under causal masking, this position can access the first-hop prefix but cannot read the subsequent r_2 token. This location is motivated by prior observations of bridge information at the first-hop position (Biran et al., 2024; Wang et al., 2024).

Let L denote the number of decoder blocks, numbered from 1 to L. For each source index $i \in \{1, \ldots, L\}$, we extract the source query's r_1-position state. We replace the recipient's corresponding state at target index $j \in \{0, \ldots, L\}$ and continue the forward pass. Target index 0 denotes the embedding output, and $j \geq 1$ denotes the output of decoder block j. Source indices $1, \ldots, L-1$ likewise denote block outputs, whereas source index L denotes the final block output after the model's final normalization. In particular, the source and target sites at index L are respectively after and before this normalization.

Only the selected position is overwritten at the intervention site; subsequent activations are computed using the modified state. We compute the full grid over all source and target indices. Figure 2 displays target indices 0 through 12.

The intervention transfers information from a different first-hop prefix while keeping the requested relations fixed. The source state cannot directly access the query's second-relation token, but it may encode information about (h, r_1) beyond the bridge entity itself.

Metric. We score the two candidate completions by teacher forcing. Each completion consists of a leading space, the answer entity name, and a terminating period. For each pair, let $P_{i,j}$ denote the model with that pair's source state injected into the recipient; we suppress the pair dependence in the notation. For a candidate answer a, let $|a|$ be its number of answer tokens. We define its mean answer-token log probability and the aggregate preference rate as

$$\begin{aligned} s_{i,j}(a \mid q') &= \frac{1}{|a|} \sum_{k=1}^{|a|} \log P_{i,j}(a_k \mid q', a_{<k}), \\ R_{i,j} &= \frac{1}{n} \sum_{p=1}^{n} \mathbf{1}\{ s_{i,j}(t_p \mid q'_p) > s_{i,j}(t'_p \mid q'_p) \}. \end{aligned} \tag{66}$$

Figure 2 reports $R_{i,j}$, the fraction of pairs for which the patched model prefers the source answer over the recipient answer under this score. We do not condition on the recipient preferring t' before intervention, so $R_{i,j}$ measures post-intervention preference rather than verified answer switches. No answers are generated for this metric.

We also compute preference for t over t' on the unpatched recipient and source queries, using the same pairs and candidate answers. The recipient rate provides the baseline without intervention,

while the source rate provides a reference for preference under the original source context. These pairwise measures are distinct from the downstream accuracy metrics in Table 3.

Varying the source and target indices independently allows us to compare different source states at the same recipient stage, and the same source state at different recipient stages. The resulting patterns characterize the transferability of first-hop information under intervention. They depend on both the information in the source state and its compatibility with the remaining recipient computation. A low preference rate therefore does not establish that bridge information is absent, and reduced transfer at late target layers does not uniquely identify when the answer was computed.