Evaluating n-Gram Novelty of Language Models Using RUSTY-DAWG 🐶

William Merrill[α,β] **Noah A. Smith**[γ,β] **Yanai Elazar**[β,γ]
[α]New York University [β]Allen Institute for AI [γ]University of Washington
willm@nyu.edu, noah@allenai.org, yanaiela@gmail.com

Abstract

How novel are texts generated by language models (LMs) relative to their training corpora? In this work, we investigate the extent to which modern LMs generate n-grams from their training data, evaluating both (i) the probability LMs assign to complete training n-grams and (ii) n-novelty, the proportion of n-grams generated by an LM that did not appear in the training data (for arbitrarily large n). To enable arbitrary-length n-gram search over a corpus in constant time w.r.t. corpus size, we develop RUSTY-DAWG, a novel search tool inspired by indexing of genomic data. We compare the novelty of LM-generated text to human-written text and explore factors that affect generation novelty, focusing on the Pythia models. We find that, for $n > 4$, LM-generated text is *less novel* than human-written text, though it is *more novel* for smaller n. Larger LMs and more constrained decoding strategies both *decrease novelty*. Finally, we show that LMs complete n-grams with lower loss if they are more frequent in the training data. Overall, our results reveal factors influencing the novelty of LM-generated text, and we release RUSTY-DAWG to facilitate further pretraining data research.[1]

1 Introduction

Despite an explosion of new applications of language models (LMs), a core question about LMs as text generators has not been fully answered: *how novel is the text they generate compared to their training corpus?* This question has both scientific value and practical relevance for LM deployment. From a scientific perspective, language understanding is often theorized as hinging on compositionality, meaning that an infinite range of meanings can be expressed by combining a small set of words or morphemes. If LMs were largely copying sentences or spans they had seen before, this would suggest they cannot compositionally generate new sentences like humans can. From a societal perspective, the novelty of LM-generated text may also be relevant to legal questions of whether copyrighted materials can be used in LM pretraining data. For instance, a lawsuit between the New York Times and OpenAI (ongoing at the time of writing) hinges on the legal ambiguity of whether including copyrighted material in training data is allowed under fair use (Klosek, 2024). Scientific evaluation of copying behavior in LMs may help guide the resolution of such questions.

In past work, McCoy et al. (2021) evaluated the novelty of text generated by sampling from small LMs, finding that small n-grams in LM-generated text are less novel than in validation text, though larger n-grams are more novel. However, McCoy et al. (2021)'s LMs were trained on WebText (40 GB; Radford et al., 2019), which is 3% of the size of the Pile (1254 GB; Gao et al., 2020), which is more representative of recent pretraining datasets. Thus, it is unclear how their conclusions would transfer to larger-scale, modern LMs.

In this work, we evaluate the n-gram generation novelty of LMs of varying sizes trained on *large-scale web data*. Specifically, we measure the proportion of generated n-grams that are novel w.r.t. the training set across many n, which we call n-novelty. Scaling the analysis of n-novelty to large corpora is challenging because measuring large-n-gram statistics over large corpora is infeasible when implemented naively. To solve this problem, we develop RUSTY-DAWG, a search tool that uses the Compacted Directed Acyclic Word Graph (CDAWG, Crochemore and Vérin, 1997; Inenaga et al., 2005) data structure for arbitrary-length n-gram matching over a corpus in *constant time* w.r.t. the corpus size and linear w.r.t the query size. While similar approaches were previously applied to genome data, we develop new algorithms for inference and extending the CDAWG with frequency information, and we are the first, to our

[1] https://github.com/viking-sudo-rm/rusty-dawg

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 14459–14473
November 12-16, 2024 ©2024 Association for Computational Linguistics

knowledge, to scale up CDAWGs to LM training data. We use RUSTY-DAWG to address the following research questions, focusing on the Pythia suite of LMs (Biderman et al., 2023) trained on the Pile:

RQ1. How novel is typical text generated by LMs compared to new human-written text?

RQ2. How do model size, decoding strategies, and prompting with training data influence the novelty of model-generated text?

RQ3. Across n-gram sizes, how does the occurrence and frequency of n-grams in the training set impact their completion loss?

Our contributions and findings are as follows:

0. We introduce **RUSTY-DAWG**, an efficient data structure based on CDAWG automata that enables unbounded-length n-gram searches in massive pretraining datasets.

1. We find **large n-grams ($n > 4$) are less novel** in LM-generated text compared to human-written text, though small n-grams ($n \leq 4$) are more novel (RQ1, Section 5.1).

2. We show that **novelty decreases with larger LMs and constrained decoding** (RQ2, Section 5.2). To an extent, prompting with training data also decreases novelty.

3. We show **LMs complete frequent training n-grams with lower loss** (RQ3, Section 6).

2 Operationalizing Novelty with n-Grams

There are different ways to measure LM generation novelty: one could assess the verbatim overlap between the text and training data (McCoy et al., 2021) or attempt to capture semantic and syntactic novelty (Shaib et al., 2024a,b). We target verbatim novelty, which is understudied at large model scale and conceivably provides an upper bound on more semantic notions of novelty (any instance of verbatim repetition likely also constitutes semantic repetition). We formalize verbatim novelty via two n-gram-based metrics: **n-novelty** and **non-novel suffix length (NNSL)**.

n-Novelty. Novelty can be evaluated at different levels of granularity. For example, while all individual tokens in a generated text will likely have occurred, it would be notable if a 100-gram from the pretraining data was generated verbatim. To capture novelty across different n-gram lengths, we follow McCoy et al. (2021) in plotting the n-novelty curve, i.e., the novelty of generated n-grams (where n varies) w.r.t. some fixed corpus C. Formally, for any text query Q (e.g., a model-generated document) we define the n-novelty rate of Q as the proportion of n-grams in Q that also occurred in C. We visualize the n-novelty curve as a function of n in Figure 1a. Intuitively, 1-novelty should be close to zero (due to the way tokenizers work), and the curve will monotonically increase with n (since substrings of a non-novel n-gram are non-novel).

NNSL. We propose a new measure of aggregate novelty across different values of n. We define NNSL at token i of Q as the length of the longest suffix of $Q[: i]$ that appeared in corpus C. We then aggregate with mean or max.

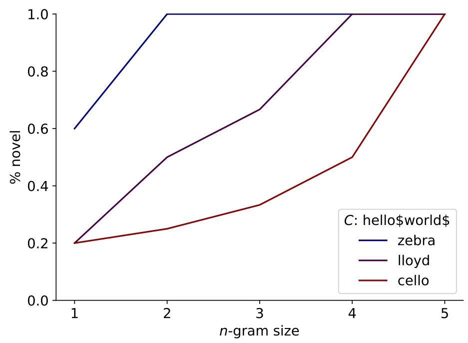
Example. Let C = `hello$world$` be a character-tokenized corpus, where `$` is a document boundary. Query $Q =$ `lloyd` has 1-gram novelty 1/5 (`y` is novel), 2-gram novelty 2/4 (`oy` and `yd` are novel), 3-gram novelty 2/3 (only `llo` is non-novel), and 4-gram novelty 2/2. The NNSL at each position is $\langle 1, 2, 3, 0, 1 \rangle$, with mean 1.4 and max 3. We intuitively demonstrate this example in Figure 1a.

3 Measuring Novelty with CDAWGs

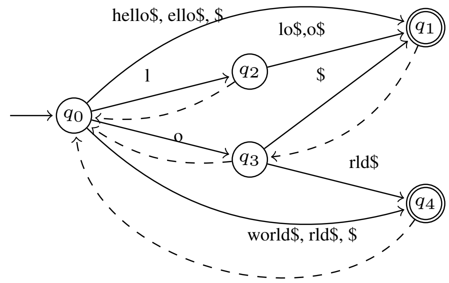
Naively computing our novelty metrics is prohibitively expensive over a large pretraining corpus like the Pile (334B tokens). To make the searches fast, we use a Compacted Directed Acyclic Word Graph (CDAWG; Crochemore and Vérin, 1997; Inenaga et al., 2005), a data structure which returns the NNSL at each position in Q against C in *constant time* (w.r.t. the size of C), and *linear time* (w.r.t. the size of Q), from which n-novelty can be computed. We describe how to compute NSSL using CDAWG in Appendix A. This constant-time querying is crucial for our application of searching the Pile. We first discuss querying CDAWGs (Section 3.1), then their memory costs (Section 3.2), their construction (Section 3.3), and our open-source implementation (Section 3.4).

3.1 Querying CDAWGs

A CDAWG is a finite-state machine built for a corpus C that acts as a rich index for C (see Figure 1b). In Appendix B.1, we propose an algorithm where a CDAWG for C can be used to efficiently answer **NNSL queries** on C:



(a) Novelty curves computed from the CDAWG in Figure 1b, labeled by their corresponding queries.



(b) CDAWG for C = `hello$world$`, where `$` is a document separator. Dashed arrows are failure arcs.

Figure 1: Illustration of CDAWG and resulting novelty curves with character-level tokenization for simplicity.

- INPUT: A string (e.g., $Q =$ `lloyd`).
- OUTPUT: NNSL at each position in Q (e.g., $L(Q) = \langle 1, 2, 3, 0, 1\rangle$) as well as the training frequencies of the largest suffixes matched at each position (e.g., $N(Q) = \langle 3, 1, 1, 0, 1\rangle$).

The algorithm follows a single arc for each token in Q, maintaining a state that encodes the largest currently matched suffix. This takes time $O(|Q|)$ with *no dependence* on $|C|$, which makes CDAWGs useful faster than suffix arrays (Carlini et al., 2023; Liu et al., 2024) for searching large corpora.

For illustration (Figure 1), we use character-level tokenization, but this process can be applied with any tokenization. The n-novelty curve, as well as all other data presented in this paper, can be computed from non-novel suffix queries.

3.2 Memory Overhead

A practical concern for an indexing data structure is its memory overhead: how many bytes does it use on a corpus of size $|C|$? The CDAWG refines the earlier Directed Acyclic Word Graph (DAWG; Blumer et al., 1984) to reduce memory overhead. A DAWG contains at most $2|C|$ states and $3|C|$ arcs (Blumer et al., 1984), which, while linear, becomes impractical for large datasets. In contrast, a CDAWG achieves $0.18|C|$ states and $0.97|C|$ arcs on the Pile. As a result, we find the CDAWG takes $\sim$50% as much memory to store as the vanilla DAWG in practice.[2] Still, the CDAWG takes $29|C|$ bytes vs. $7|C|$ for a suffix array, illustrating a time/space tradeoff between the two approaches.

Another factor that affects memory overhead is the choice of graph representation. We implemented the edge list for a node with an AVL tree to make transitions very fast, but at the cost of some memory overhead. Further details about the graph representation, memory overhead, and potential improvements can be found in Appendix B.3.

3.3 Building CDAWGs

The naive way to build a CDAWG would involve enumerating all span in a corpus in quadratic time, which is infeasible for large corpora. Luckily, more refined algorithms for building DAWGs and CDAWGs were developed that process each token in the corpus left to right, taking linear time overall (Blumer et al., 1984; Crochemore and Vérin, 1997; Inenaga et al., 2005). We implement Inenaga et al. (2005)'s linear-time algorithm.

In Appendix B.2, we propose a graph-traversal algorithm that can be used to add n-gram frequency information to the CDAWG after it has been created, stored at the states. Since edges dominate the memory overhead of the CDAWG, this only minimally increases the space overhead but allows us to return the count associated with matched n-grams, which we expect to be useful in many applications.

3.4 RUSTY-DAWG Library

While there are some pre-existing open-source libraries for DAWGs,[3] we did not find a scalable open-source implementation of CDAWGs. To facilitate our research and other applications of CDAWGs to large text corpora, we implemented RUSTY-DAWG, a modern Rust library for building and using DAWGs and CDAWGs, which we open-sourced at: `https://github.com/viking-sudo-rm/rusty-dawg`. RUSTY-DAWG

[2] A CDAWG arc is larger than a DAWG arc. Hence, the CDAWG memory overhead is reduced by 50% (and not more) despite a larger reduction in the number of states and arcs.

[3] `https://github.com/elake/SuffixAutomaton`

provides a memory-efficient implementation of CDAWGs, their construction algorithm, and our algorithms for populating them with frequency information and answering NNSL queries. It also provides useful features like the ability to store the CDAWG in RAM or on disk (and switch between these modes). RUSTY-DAWG enabled us to build, to our knowledge, the largest CDAWG every created. See Appendix B.4 for more details.

4 Experimental Setup

4.1 Building a CDAWG on the Pile

We focus our study on the copying behavior of the eight Pythia models (Biderman et al., 2023) trained on the Pile (Gao et al., 2020). The Pile contains many kinds of text, including web text, books, code, and email communication. We build our RUSTY-DAWG on the non-deduplicated version using the GPT-NeoX (Black et al., 2022) tokenization used by Pythia, under which it contains 334B tokens.

To parallelize building RUSTY-DAWG, we divide the Pile documents into 30 shards and build a CDAWG on each 11B-token shard on a different cloud machine. Each of the 30 created CDAWGs has 2B states and 11B arcs, taking 327 GB total memory. We store this in RAM during building. At inference time, we keep the CDAWG shards on disk and execute NNSL queries on each of the 30 CDAWGs in parallel. We aggregate NNSL (by taking the max) and counts returned (by summing at maximum suffix lengths) to exactly simulate the output of a single CDAWG.

4.2 Generating Text from LMs

We evaluate the generation novelty of the Pythia models (Biderman et al., 2023), which were trained on the Pile (Gao et al., 2020) at different sizes up to 12B parameters. We consider two setups, (1) generating unmprompted texts, and (2) generating prompted texts, for which we sample 500 documents from the Pile validation set (trimmed to 1,000 tokens). In each setup we generate 500 documents of 1,000 tokens from each LM. We vary the model size (from 70M to 12B, 8 models in total) and decoding strategy, sweeping different parameters for top-p (nucleus sampling; Holtzman et al., 2020), top-k (Fan et al., 2018), temperature, and greedy beam search. Unless otherwise indicated, we use Pythia-12B and standard sampling with an unconditioned prompt as defaults. We pass each generated text through the CDAWG to compute the NNSL at each position (cf. Section 3), from which we compute the n-novelty curves.

4.3 Novelty Baselines

For small n, some n-grams will likely be repeated between a document and a large corpus by random chance. For large n this probability will decrease rapidly. Thus, to evaluate the novelty of LM generations, it is necessary to establish a baseline n-novelty curve. We consider two such baselines:

Validation Text. Following McCoy et al. (2021), we use the novelty of text in the Pile's validation set as a baseline. If n-grams of a certain size are less novel in generated text compared to validation text, the LM is generating pretraining n-grams more commonly than expected for new documents from the pretraining distribution. This suggests the LM is copying from its pretraining corpus.

Text After Pile Date Cutoff. The novelty of validation text may be artificially low if the training distribution contains duplicated documents (Lee et al., 2022a). To account for this, we filter text from Dolma (Soldaini et al., 2024) that was written after the Pile collection cutoff. Specifically, the two domains we use are Reddit and scientific texts (Pes2o; Soldaini and Lo, 2023), both of which are in-distribution for the Pile. Thus, we expect this baseline to represent natural overlap for human-written text without contamination. We report the n-novelty curve fit on both domains from Dolma together, though qualitatively we observe that the curve looks similar within each domain.

For both the Pile and Dolma, we sample 500 validation documents and truncate each document to the first 1,000 tokens.

5 Novelty of LM-Generated Text

5.1 Novelty vs. Human-Written Text

We now compare the novelty of typical LM-generated text against human text (RQ1). As such, we report novelty metrics for two human-written text baselines: validation text and Dolma documents written after the Pile cutoff.

Validation Baseline. Figure 2 shows that the validation n-novelty curve is very low across n. 2.4% of the 1,000-token validation documents are *exactly matched* somewhere in the Pile training set. 13.6% share a 100-gram with the training data, and 25.0% share a 50-gram. This suggests contamination, since we expect natural large-n-gram overlap should be vanishingly unlikely.

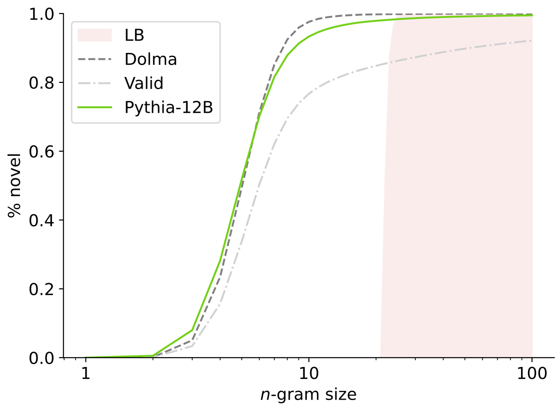


Figure 2: n-novelty curve for Pythia-12B with naive sampling. Compared to Dolma, LM-generated text is more novel for $n > 4$ and slightly less novel for $n \leq 4$. The gap between the dark gray Dolma curve and the green Pythia-12B curve quantifies the novelty difference. LM-generated text is more novel than the Pile validation set across n-gram sizes due to contamination.

To formally test this, we derive a lower bound on n-novelty assuming most next tokens are nondeterministic (Appendix C).[4] Under this assumption, the n-novelty curve for non-contaminated data should not enter the red region in Figure 2, i.e., almost all 23+-grams should be novel. The validation curve (but not Dolma) enters this region, suggesting many contaminated n-grams in the validation text. Thus, we turn to Dolma as a better representation of uncontaminated human-written text.

Dolma Baseline. Figure 2 shows that n-grams of size $n > 4$ are less novel in generated text compared to Dolma text, whereas n-grams of size $n \leq 4$ (median length) are slightly more novel. For instance, 8% of Pythia bigrams are novel (vs. 5% for Dolma), while 93% of Pythia 10-grams and 99% of 100-grams are novel (vs. 98% and 100%). This disagrees with McCoy et al. (2021)'s findings for small LMs trained on 40 GB of text, where LMs were less novel for small n-grams (2-3% bigram novelty for LMs vs. 6% baseline novelty). One explanation for the difference may be the model and data scale, motivating us to more closely analyze the impact of model size on novelty in Section 5.2.

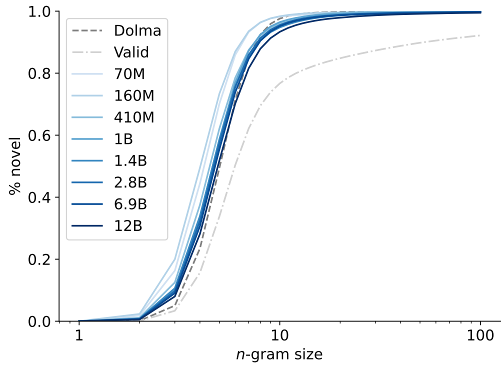
Examples of Copied n-Grams. We find that many non-novel n-grams generated by Pythia-12B are pieces of licenses and boilerplate code. For example, Pythia-12B generates a 64-gram with 45K occurrences in the Pile that starts:

> *//*
> *// Licensed under the Apache License, Version 2.0 (the "License");*
> *// you may not use this file except in compliance with the License...*

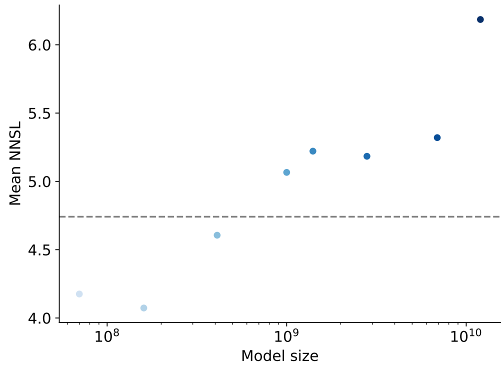
Another generated 64-gram (with 213K occurrences in the Pile) imports Linux libraries:

> *...#include <sound/core.h>*
> *#include <sound/pcm.h>*
> *#include <sound/soc.h>...*

[4] We make the possibly strong assumption that 90% of tokens per n-gram have entropy $\ell \geq 1.8$ bits/token based on the best achieved Pile losses (Du et al., 2022). See Appendix C.



(a) n-novelty curves across model sizes.



(b) Mean NNSL across model sizes.

Figure 3: Both n-novelty and mean NNSL suggest larger LMs generate less novel text than smaller LMs.

5.2 Impact of Model Size and Decoding

Having explored the novelty of LM-generated text compared to human-written text, we assess the factors that influence the generation novelty of LMs (RQ2). We compare n-novelty curves for varying model sizes, decoding strategies, and different prompt lengths from the training data. The default values of each variable (when sweeping other variables) are 12B, standard sampling, and a prompt length of 0.

Setup	Param	Mean	Max
Baseline	Validation	29.94	1,000
	Dolma	4.74	66
Size	70M	4.18	187
	160M	4.07	207
	410M	4.61	191
	1B	5.07	270
	1.4B	5.22	225
	2.8B	5.18	322
	6.9B	5.32	198
	12B	6.19	376
Prompt	1	5.83	624
	10	6.21	393
	100	7.56	976

Table 1: NNSL results for human-written text baselines and different model sizes and prompt lengths.

Decoding	Param	Mean	Max
Baseline	Validation	29.94	1,000
	Dolma	4.74	66
Top-p	0.85	15.02	992
	0.9	8.85	1000
	0.95	9.69	902
Top-k	20	11.34	507
	80	9.24	580
	160	8.17	386
Temperature	0.5	14.22	983
	0.85	10.18	969
	0.9	11.05	1,000
	0.95	6.55	418
	1.05	5.08	313
	1.1	4.34	375
Beam	8	192.03	408
	4	9.17	18
	1	8.40	19

Table 2: NNSL results for Pythia-12B with different decoding strategies. Across strategies, more constrained decoding leads to less novel text.

Larger LMs are Less Novel. Figure 3a shows that, across n, n-grams are less novel for larger LMs than for smaller LMs. Similarly, Figure 3b shows that the mean NNSL increases linearly with log model size. Both metrics suggest that larger LMs are *less novel* than smaller LMs across all n-gram sizes. This may indicate that larger LMs have more capacity to memorize n-grams from training.

Decoding Constraints Decrease Novelty. Prior work with small LMs and corpora suggests decoding choices could influence generation novelty (McCoy et al., 2021). In particular, we expect more constrained decoding to decrease novelty (Liu et al., 2024). To evaluate this, we generate text with top-p, top-k, temperature (including greedy), and greedy beam search decoding setups, varying the parameter that constraints generation in each case. We hypothesize that the parameter choices that more constrained will result in lower generation novelty.

Indeed, Figure 4 shows that constrained decoding reduces n-novelty. The constrained decoding curves are consistently below the Dolma baseline, and, for small n, even below the validation baseline. The least n-novel approaches are low-temperature decoding and beam search. For 10-grams, temperature 0.5 reaches 71% novelty and temperature 0 reaches 69% novelty, while for 100-grams, temperature 0.5 reaches 98% and temperature 0 reaches 100%. Increasing beam size decreases novelty, with beam size 8 remaining near 0% novelty even up to 100-grams. With beam size 8, the LM deterministically generates a single document containing a 408-gram from training (see Appendix D for beam size results with nondeterministic conditioned generation). As we show in Table 2, temperature ≤ 0.9 and top-$p \leq 0.95$ dramatically increase both the mean NNSL and the max. Both novelty metrics suggest that constrained decoding reduces generation novelty.

Long Training Prompts Slightly Decrease Novelty. To evaluate the impact of prompting with training data, we prompt the model with p tokens from the beginning of a training document before generating 1,000 additional tokens. We then evaluate the n-novelty curve for these 1,000 tokens. Qualitative inspection reveal that the novelty curves look almost identical, independently of the prompt length p. However, the NNSL statistics (Table 1) tell a more subtle story: the median NNSL remains unchanged, whereas the mean increases from 6.19 to 7.56 with 100 prompt tokens. This suggests that, while most n-grams do not become more novel when a prompt is given, the longest non-novel n-grams are longer when a longer prompt is given.

6 Impact of n-Gram Training Frequency

Finally, regarding RQ3, we test whether, at inference time, LMs assign higher probability to n-grams from training, and how this interacts with training frequency. We define the **mean comple-**

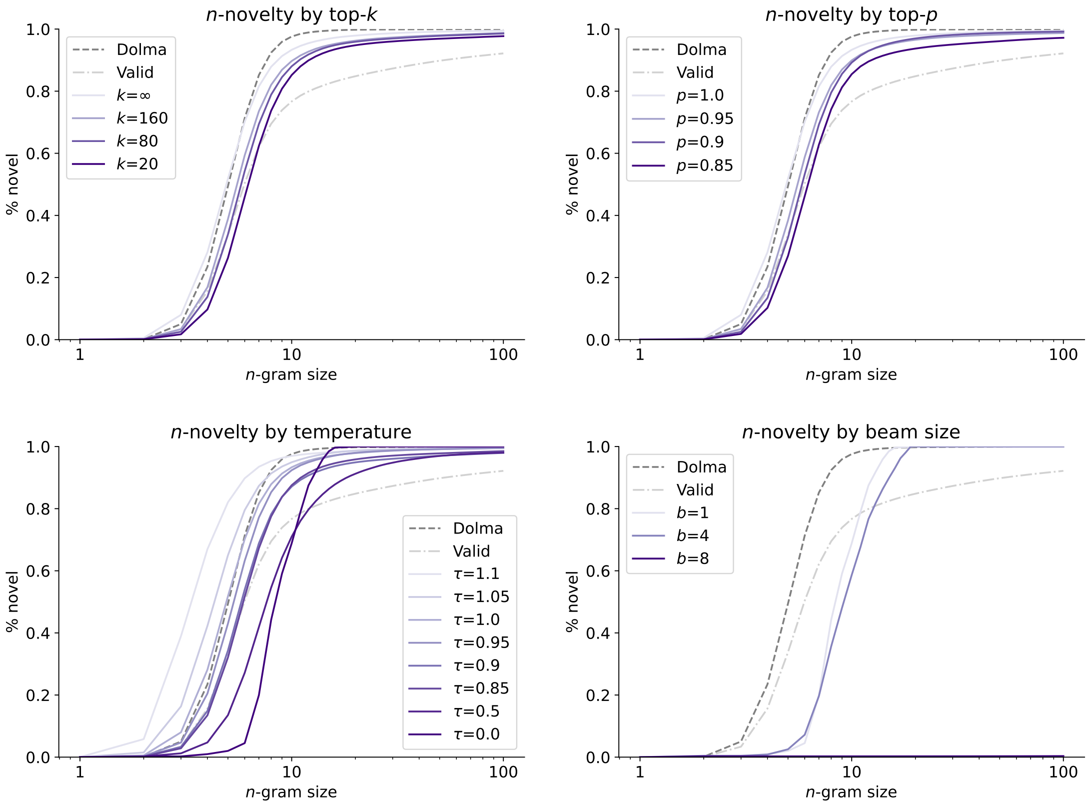


Figure 4: Impact of decoding choices (top-k, top-p, temperature τ, and beam size with $\tau = 0$) on n-novelty. Less stochastic (darker) decoding choices *decrease novelty*; temperature and beam size have the strongest effect.

tion loss of $x_1 \cdots x_n$ as the average probability the LM assigns to x_n when it occurs in validation text:

$$\hat{\ell}(x) = \frac{1}{|\mathcal{V}_x|} \sum_{i \in \mathcal{V}_x} p_{\mathrm{LM}}(v_i \mid v_{1:i-1}),$$

where $\mathcal{V}_x = \{i : v_{i+1-n:i} = x\}$. This captures the LM's sensitivity to training n-grams in a way that is independent of the specific sampling choices made when decoding from the LM. It also captures use cases of LMs where the LM is used to assign probabilities to strings rather than as a text generator, such as in multiple-choice question answering like MMLU (Hendrycks et al., 2021) or evaluation of noun-verb agreement (Marvin and Linzen, 2018).

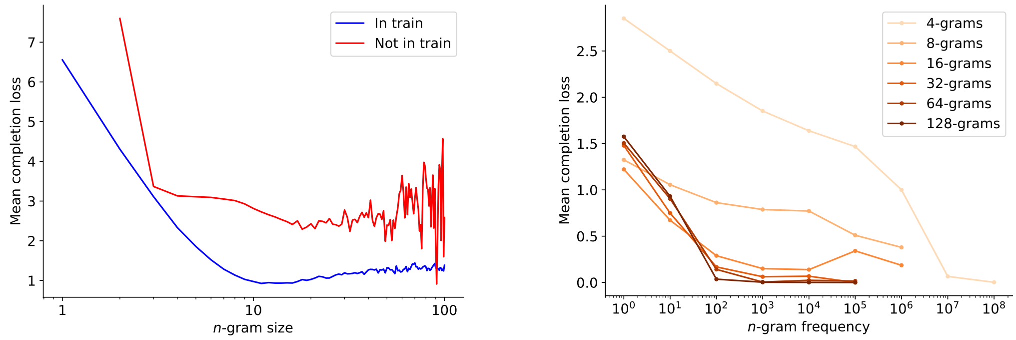
Method. We sample 5,000 documents of 1,000 tokens each, from the Pile validation set. We compute the per-token loss using Pythia-12B and use the CDAWG to find the non-novel suffixes at each position. For each n, we find tokens in the validation data that fall into two categories:

- **In Train:** The n-gram ending at the token occurred in the training data.
- **Not in Train:** The n-gram ending at the token did not occur in the training data, but the $(n-1)$-gram ending at the previous token did.

We then compute the mean completion loss across all tokens in each condition with the same value of n, and plot this mean loss as a function of n. This quantity measures the surprisal of the LM when completing n-grams, with the two conditions differentiating whether the correct n-gram completion appeared in the training data. For the n-grams in the training data, we also investigate how their frequency affects completion loss.

Training n-Grams are Easier to Complete. Figure 5a shows that, across n-gram sizes, the completion loss for n-grams from the training set is smaller than for n-grams not in the training set (concretely, for n-grams above size 10, the completion loss is roughly 50% less when the n-gram was in the training set). For $n > 80$, the loss curve for n-grams not in training becomes noisy, reflecting the rarity of such n-grams. These results suggest that Pythia-12B is upweighting tokens that complete n-grams from pretraining.[5] This finding potentially

[5]While these results may be confounded (training n-grams may be easier to complete for other reasons besides their occurrence in the training set), we believe this is not a significant



(a) Completion loss of Pythia-12B on n-grams in validation text based on whether the n-grams occurred in training. Across n-gram sizes, Pythia-12B assigns lower loss to n-grams seen during training.

(b) Completion loss as a function of n-gram training frequency for different n-gram sizes. Across n-gram sizes, more frequent n-grams have lower loss (with larger n being easier to predict).

Figure 5: n-gram completion loss based on presence in train and frequency.

explains why more constrained decoding decreases novelty: while LMs assign probability to complete training n-grams, their next-token prediction with standard sampling also places a lot of probability mass on other tokens. Thus, training n-grams may not always get generated. However, the finding that training n-grams are upweighted in terms of probability suggests that pruning probability mass on other tokens (as approaches like top-p or top-k do) would cause even more training n-grams to be generated, as found in Section 5.2.

Frequent n-Grams are Easier to Complete. Figure 5b shows that, across sizes, n-grams that are more frequent in the training data are easier for Pythia-12B to complete, implying LM predictions are sensitive to training data frequency effects. This is particularly relevant when specific token continuations are compared to assess multiple choice answers: e.g., `a)`, `b)`, `c)`, and `d)` for MMLU evaluation (Hendrycks et al., 2021), or comparing `is`/`are` to assess noun-verb agreement competence (Marvin and Linzen, 2018). Table 3 shows that the Pile frequency of these continuations are not uniform. Combined with Figure 5b, this suggests evaluating LMs by comparing these tokens may be susceptible to pretraining frequency effects.

In summary, these results suggest that LMs are more likely to memorize and copy n-grams with higher frequency in the pretraining data.

issue. See Appendix D for analysis of a possible confounder: NNSL by token index.

`a)`	`b)`	`c)`	`d)`
$\underline{2.5 \times 10^7}$	2.3×10^7	2.1×10^7	1.1×10^7
	`is`	`are`	
	$\underline{1.4 \times 10^8}$	2.9×10^7	

Table 3: n-gram frequencies in the Pile computed by CDAWG. `a)` is more frequent than other options, and `is` is more frequent than `are`. Combined with Figure 5b, this suggests evaluations that use continuation probabilities may be susceptible to pretraining frequency effects.

7 Related Work

7.1 Methods for Accessing Text Corpora

Data is becoming an important factor for understanding LM behavior (Elazar et al., 2024). As the scale of pretraining datasets continues to increase, naive search through these large datasets does not scale. As such, we need clever algorithms and data structures to interact with and study huge datasets.

McCoy et al. (2021), the first work to study the generation novelty of LMs, trained on Wikitext-103 (<1 GB) and WebText (40 GB). At this small data scale, they matched n-grams by brute-force enumeration, which would not be feasible today with the Pile (1254 GB) or larger datasets. In contrast, Elazar et al. (2024) use an elastic search index based on an inverted index that allows a to search a corpus which depends on the number of documents in the corpus, making it much slower then our approach. Carlini et al. (2023); Liu et al. (2024) use a suffix array (Manber and Myers, 1990), allowing queries in logarithmic time w.r.t. corpus size. Another data structure previously used in the setting of text generation with retrieval is the FM-index (Ferragina and Manzini, 2000; Bevilacqua et al., 2022), a compressed suffix array.

In this work, we use a CDAWG (Crochemore and Vérin, 1997; Inenaga et al., 2005), which is a refinement of the earlier DAWG (Blumer et al., 1984), and part of a larger family of "$*$DAWG" indices (Takagi et al., 2017; Inenaga, 2024). $*$DAWGS use more memory than suffix arrays but support faster membership and NNSL queries (cf. Section 3). $*$DAWGs also naturally allow computing arbitrary-length n-gram probabilities (Liu et al., 2024), which could be useful for retrieval language modeling applications.

7.2 Memorization, Contamination, and Generalization

The increased use of LMs has raised concerns about memorization artifacts that might limit their generalization potential. For instance, Bender et al. (2021) draw a parallel of LMs to "stochastic parrots" that memorize and mimic their training data.

Memorization has been carefully studied and quantified (Zhang et al., 2021; Kandpal et al., 2022; Lee et al., 2022b; Magar and Schwartz, 2022; Carlini et al., 2023; Ippolito et al., 2023) and is often framed as a concerning property of model behavior. On the other hand, other works claim that memorization is integral for generalization (Feldman, 2020; Feldman and Zhang, 2020; Chatterjee, 2018). In this work, we do not take a stance on the importance or dangers of memorization, but rather quantify the novelty of LM-generated text vs. human text and investigate how different parameters affect novelty. In contrast to much previous work on memorization, we also focus on the novelty of typical text rather than text elicited in adversarial settings (Carlini et al., 2023; Ippolito et al., 2023).

Like McCoy et al. (2021), we focus on generation novelty rather than quality, and we are interested in the effect of different variables such as model size, and decoding strategies on the generation novelty. Due to the CDAWG, we are able to scale our analysis to larger datasets than McCoy et al. (2021). In addition to text diversity, Shaib et al. (2024a,b) investigated the diversity of generated part-of-speech sequences rather than texts, as an abstract measurement over the raw texts. Lesci et al. (2024) consider the effect of data ordering on LM memorization. In future work, it would be possible to extend our CDAWG analysis to look at data ordering by building the CDAWG on documents in the same ordering as a pretraining run and performing analysis at different checkpoints of the partial CDAWG.

8 Discussion

A Note on Generation Quality. In this work, we focus on quantifying n-gram novelty in model generations with respect to the model's training data. One confounding factor that may influence the *cause* of novelty in such generations is the quality of the text. For example, consider a model that generates the sequence "n-gram n-gram n-gram n-gram." This sequence is of low quality and is unlikely to appear in a corpus (it does not appear in Dolma), making it a novel 4-gram. Investigating how n-gram novelty interacts with text quality is left for future work.

Other CDAWG Use Cases. Several recent studies have explored memorization during training (Chang et al., 2024; Lesci et al., 2024). These studies performed interventions on the data used for training LMs at different checkpoints to study the effect of model size, training step, etc. on memorization. Since CDAWG is constructed sequentially, future research could leverage this property to mirror the order in which the data was presented to the model during training. This would enable the use of CDAWGs for studying how novelty evolves over the course of pretraining for a specific data ordering.

9 Conclusion

We introduce RUSTY-DAWG, an efficient index for finding *unbounded length* n-gram overlap against a pretraining corpus whose runtime is independent of the corpus size. Using RUSTY-DAWG, we show that, at large n, Pythia generates less novel n-grams than novel human-written text. We also find that increasing model size, constrained decoding (e.g., with temperature 0), or prompting with training data decrease novelty — in particular, low temperature has a dramatic effect. Finally, more frequent training n-grams are completed by LMs with *lower loss*, suggesting LMs are more prone to memorizing frequent n-grams. We hope RUSTY-DAWG enables further analysis of pretraining data as well as decontamination (Magnusson et al., 2023) and retrieval language modeling (Khandelwal et al., 2020; Liu et al., 2024) research.

Limitations

When evaluating novelty, we focus on verbatim n-gram novelty rather than evaluating semantic novelty, which would be harder to operationalize. Our analysis focuses on the non-deduplicated Pile, a primarily English dataset. There are many variables about data curation or LM training that could affect generation novelty beyond the ones we have considered, which could be explored using similar methodology in future work. Finally, as discussed in Appendix B.3, one challenge with deploying the CDAWG is the memory overhead, though we believe this can be optimized in future work.

Acknowledgements

We thank Ananya Jha, Rodney Kinney, Dave Wadden, and Pete Walsh for their engineering contributions to RUSTY-DAWG during the AI2 hackathon and Michal Guerquin, Johann Dahm, and the Beaker team for assistance running RUSTY-DAWG on AI2 infrastructure. We thank Shunsuke Inenaga for guidance on the CDAWG data structure and Cyril Allauzen, Jiacheng Liu, and Vishakh Padmakumar for advice and feedback on this paper.

References

Cyril Allauzen. 2023. Weighted finite automata with failure transitions: Algorithms and applications. In *Proceedings of 16th edition of the International Conference on Grammatical Inference*, volume 217 of *Proceedings of Machine Learning Research*, pages 6–6. PMLR.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA. Association for Computing Machinery.

Michele Bevilacqua, Giuseppe Ottaviano, Patrick Lewis, Scott Yih, Sebastian Riedel, and Fabio Petroni. 2022. Autoregressive search engines: Generating substrings as document identifiers. *Advances in Neural Information Processing Systems*, 35:31668–31683.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, Usvsn Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, pages 95–136, virtual+Dublin. Association for Computational Linguistics.

A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, and R. McConnell. 1984. Building the minimal dfa for the set of all subwords of a word on-line in linear time. In *Automata, Languages and Programming*, pages 109–118, Berlin, Heidelberg. Springer Berlin Heidelberg.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. 2023. Quantifying memorization across neural language models. In *The Eleventh International Conference on Learning Representations*.

Hoyeon Chang, Jinho Park, Seonghyeon Ye, Sohee Yang, Youngkyung Seo, Du-Seong Chang, and Minjoon Seo. 2024. How do large language models acquire factual knowledge during pretraining? *arXiv preprint arXiv:2406.11813*.

Satrajit Chatterjee. 2018. Learning and memorization. In *International conference on machine learning*, pages 755–763. PMLR.

Maxime Crochemore and Renaud Vérin. 1997. *On compact directed acyclic word graphs*, pages 192–211. Springer Berlin Heidelberg, Berlin, Heidelberg.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.

Yanai Elazar, Akshita Bhagia, Ian Helgi Magnusson, Abhilasha Ravichander, Dustin Schwenk, Alane Suhr, Evan Pete Walsh, Dirk Groeneveld, Luca Soldaini, Sameer Singh, Hanna Hajishirzi, Noah A. Smith, and Jesse Dodge. 2024. What's in my big data? In *The Twelfth International Conference on Learning Representations*.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

Vitaly Feldman. 2020. Does learning require memorization? a short tale about a long tail. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 954–959.

Vitaly Feldman and Chiyuan Zhang. 2020. What neural networks memorize and why: Discovering the long tail via influence estimation. *Advances in Neural Information Processing Systems*, 33:2881–2891.

P. Ferragina and G. Manzini. 2000. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398.

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

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Shunsuke Inenaga. 2024. Linear-size suffix tries and linear-size cdawgs simplified and improved. *Preprint*, arXiv:2401.04509.

Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, Setsuo Arikawa, Giancarlo Mauri, and Giulio Pavesi. 2005. On-line construction of compact directed acyclic word graphs. *Discrete Applied Mathematics*, 146(2):156–179. 12th Annual Symposium on Combinatorial Pattern Matching.

Daphne Ippolito, Florian Tramer, Milad Nasr, Chiyuan Zhang, Matthew Jagielski, Katherine Lee, Christopher Choquette Choo, and Nicholas Carlini. 2023. Preventing generation of verbatim memorization in language models gives a false sense of privacy. In *Proceedings of the 16th International Natural Language Generation Conference*, pages 28–53, Prague, Czechia. Association for Computational Linguistics.

Nikhil Kandpal, Eric Wallace, and Colin Raffel. 2022. Deduplicating training data mitigates privacy risks in language models. In *International Conference on Machine Learning*, pages 10697–10707. PMLR.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*.

Katherine Klosek. 2024. Training generative ai models on copyrighted works is fair use.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022a. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445, Dublin, Ireland. Association for Computational Linguistics.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022b. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445.

Pietro Lesci, Clara Meister, Thomas Hofmann, Andreas Vlachos, and Tiago Pimentel. 2024. Causal estimation of memorisation profiles. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15616–15635, Bangkok, Thailand. Association for Computational Linguistics.

Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. 2024. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. *arXiv preprint arXiv:2401.17377*.

Inbal Magar and Roy Schwartz. 2022. Data contamination: From memorization to exploitation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 157–165.

Ian Magnusson, Akshita Bhagia, Valentin Hofmann, Luca Soldaini, Ananya Harsh Jha, Oyvind Tafjord, Dustin Schwenk, Evan Pete Walsh, Yanai Elazar, Kyle Lo, Dirk Groenveld, Iz Beltagy, Hanneneh Hajishirz, Noah A. Smith, Kyle Richardson, and Jesse Dodge. 2023. Paloma: A benchmark for evaluating language model fit. *technical report*.

Udi Manber and Gene Myers. 1990. Suffix arrays: a new method for on-line string searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, page 319–327, USA. Society for Industrial and Applied Mathematics.

Rebecca Marvin and Tal Linzen. 2018. Targeted syntactic evaluation of language models. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202, Brussels, Belgium. Association for Computational Linguistics.

R. Thomas McCoy, Paul Smolensky, Tal Linzen, Jianfeng Gao, and Asli Celikyilmaz. 2021. How much do language models copy from their training data? evaluating linguistic novelty in text generation using raven. *Preprint*, arXiv:2111.09509.

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

Chantal Shaib, Joe Barrow, Jiuding Sun, Alexa F Siu, Byron C Wallace, and Ani Nenkova. 2024a. Standardizing the measurement of text diversity: A tool and a comparative analysis of scores. *arXiv preprint arXiv:2403.00553*.

Chantal Shaib, Yanai Elazar, Junyi Jessy Li, and Byron C. Wallace. 2024b. Detection and measurement of syntactic templates in generated text. *arXiv preprint 2407.00211*.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. 2024. Dolma: An Open Corpus of Three Trillion Tokens for Language Model Pretraining Research. *arXiv preprint*.

Luca Soldaini and Kyle Lo. 2023. peS2o (Pretraining Efficiently on S2ORC) Dataset. Technical report, Allen Institute for AI. ODC-By, `https://github.com/allenai/pes2o`.

Takuya Takagi, Keisuke Goto, Yuta Fujishige, Shunsuke Inenaga, and Hiroki Arimura. 2017. Linear-size cdawg: New repetition-aware indexing and grammar compression. In *String Processing and Information Retrieval*, pages 304–316.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2021. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115.

A Computing n-Novelty from NNSL

The direct output of the CDAWG is L_Q, the NNSL vector across each position in Q. We now describe how to compute the n-novelty curve from $L(Q)$. First, we define $c(n)$ as the the number of times n occurred in $L(Q)$:

$$c(n) = \sum_{\ell \in L(Q)} \mathbb{1}[n = \ell].$$

Next, the number of novel n-grams in Q is

$$\left(\sum_{n' < n} c(n) \right) - (n-1).$$

The total number of n-grams in Q is $|Q| - (n-1)$. Thus, the n-novelty is

$$n\text{–novelty}(Q) = \frac{\left(\sum_{n' < n} c(n) \right) - (n-1)}{|Q| - (n-1)}.$$

This can be extended to multiple documents by summing the numerator and denominator across documents before dividing.

B CDAWG Details

B.1 Querying the CDAWG

Algorithm 1 implements an NNSL query by greedily passing Q through the CDAWG one token at a time. We track the current state, any intermediate progress along an arc represented by indices $\langle \alpha, \gamma \rangle$ for a span in C, and the currently matched length. If no progress can be made along an arc by the next token, a failure arc (Allauzen, 2023) is followed to back off until a state with a defined transition is found (or to $\emptyset$ if no such state exists). If some partial progress is matched along an arc, that progress must be matched at the arc out of the state backed off to as well. We refer to this as an *implicit failure transition*, denoted by $\phi(q, \langle \alpha, \omega \rangle)$.

B.2 Populating Counts

We build the CDAWG according to Figure 17 of Inenaga et al. (2005). The final post-processing step we add is to populate the counts in the CDAWG via a depth-first traversal (cf. Algorithm 2). The idea is that the CDAWG represents the frequency of a string x in C by the number of paths from the node reached by x to a sink node. Further, the frequency of each node is the sum of the frequencies of its children. Thus, we can populate all the counts in the CDAWG via a depth-first traversal of its nodes, which takes time $O(|C|)$.

Algorithm 1: NNSL query with CDAWG

Data: CDAWG G with source q_0
Input: query Q
Output: NNSL vector L_Q and counts N_Q, emitted pairwise

```
s.q ← q_0;
s.⟨α, ω⟩ ← ⟨0, 0⟩;
s.ℓ ← 0;
for token σ of Q do
    s ← trans(s, σ);
    emit ⟨s.ℓ, count(s.q)⟩;
fn trans(s, σ):
    if s.q = ∅ then
        s.q ← q_0;
        s.⟨α, ω⟩ ← ⟨0, 0⟩;
        s.ℓ ← 0;
    else if α = ω then
        q′ ← target of completed arc;
        if e ← σ-edge out of q′ then
            s.q ← q′;
            s.⟨α, ω⟩ ← weight of e;
            s.ℓ ← s.ℓ + 1;
        else
            s.q ← φ(q′, ⟨α, ω⟩);
            s ← trans(s, σ);
    else
        σ′ ← token s.α of C;
        if σ = σ′ then
            s.α ← s.α + 1;
            s.ℓ ← s.ℓ + 1;
        else
            s.q ← φ(q, ⟨α, ω⟩, ℓ);
            s ← trans(s, σ);
    return s;
```

B.3 Graph Representation

An important detail for the memory usage of the CDAWG is it is represented as a graph. We represent the graph as a list of nodes and a list of edges. The edges at each node are represented by a binary AVL tree, which means the arc labelled by token $\sigma \in \Sigma$ can be found in $O(\log|\Sigma|)$ time. However, this representation means each edge takes 26 bytes (with 5 byte pointers), which leads to an overall size of $29|C|$ for the CDAWG. This is roughly $4\times$ larger than the corresponding suffix array, meaning there is a time/space tradeoff between the two approaches. We believe the memory overhead factor of the CDAWG could be significantly optimized by refining this graph representation in future work.

Algorithm 2: Add counts to CDAWG

```
Data: CDAWG G with source q_0
create stack S;
push ⟨OPEN, q_0⟩ onto S;
while ⟨o, q⟩ ← pop from S do
    if o = OPEN then
        if count(q) > 0 then
            continue;
        count(q) ← 1;
        push ⟨CLOSE, q⟩ onto S;
        for child q' of q do
            push ⟨OPEN, q'⟩ onto S;
    else
        count(q) ← 0;
        for child q' of q do
            add count(q') to count(q);
```

The memory overhead of RUSTY-DAWG could be further reduced by implementing recent improvements of the CDAWG such as the linear-size CDAWG (LCDAWG; Takagi et al., 2017) and simplified LCDAWG (simLCDAWG; Inenaga, 2024).

B.4 RUSTY-DAWG Library

DAWGs and CDAWGs can be stored in either RAM or disk to accomodate different resource constraints (building and inference are faster in RAM, but for very large datasets, using disk may be preferable due to resource constraints, especially for inference). Rust was chosen as a language so runtime and memory overhead could be optimized, though we also created Python bindings for easy integration with machine learning workflows. All experiments in the paper were carried out using the Python bindings to access a CDAWG built with the RUSTY-DAWG library.

While building our CDAWGs for the Pile, we stored them in RAM to make the process faster. This took 24 hours using 30 Google Cloud `n2-highmem-48` machines. At inference time, we transferred the CDAWGs to disk. The RUSTY-DAWG inference speed depends mainly on the speed of the RAM or disk used to store the data structure since inference is bottlenecked by memory accesses. We used relatively cheap `n2-standard-16` machines with low IO speed (15k IOPS) and observed between 100-1000 tokens per second, which sufficed for our experiments. Since memory accesses the bottleneck, the speed could be improved significantly just by using machines with faster disk (or RAM).

C Lower Bound on Novelty Without Duplication

Our theoretical lower bound baseline is based on the idea that the next token is fundamentally nondeterministic, and, therefore, long n-gram spans should be unlikely.

C.1 Warmup: Always Nondeterministic Case

Let $\mathbb{1}[\cdot]$ be an indicator, and we will use subscripts (e.g., X_i or d_i) to indicate elements of strings.

Say that we sample a corpus C of strings from some distribution p and then denote by $\mathcal{D}_n$ the set of all n-grams in C. We then let X be a random string of length n sampled from p. We say that X is n-novel if $X \notin \mathcal{D}_n$ and we are interested in analyzing this probability. This event is the complement of the events where X is any particular n-gram from C, so its probability is

$$p(X\ n\text{−novel}) = 1 - p\left(\bigvee_{d \in \mathcal{D}_n} \bigwedge_{i=1}^{n} \mathbb{1}[X_i = d_i] \right).$$

By the union bound,

$$\begin{aligned} p(X\ n\text{−novel}) &\geq 1 - \sum_{d \in \mathcal{D}_n} p\left(\bigwedge_{i=1}^{n} \mathbb{1}[X_i = d_i] \right) \\ &= 1 - \sum_{d \in \mathcal{D}_n} \prod_{i=1}^{n} p(X_i = d_i \mid X_{<i}). \end{aligned}$$

Assume p is always nondeterministic at every position, so there is some $q < 1$ such that, for all i and x,

$$p(X_i = x_i \mid X_{<i} = x_{<i}) \leq q. \tag{1}$$

Then it follows that

$$\begin{aligned} p(X\ n\text{−novel}) &\geq 1 - \sum_{d \in \mathcal{D}_n} \cdot q^n && \text{(by 1)} \\ &= 1 - |\mathcal{D}_n| \cdot q^n \\ &\geq 1 - |C| \cdot q^n. \end{aligned}$$

Finally, recasting in terms of negative log-likelihood $\ell = -\log q > 0$, we get

$$p(X\ n\text{−novel}) \geq 1 - |C| \cdot \exp(-n\ell).$$

A first observation here is that $p(X\ \ n\text{−novel})$ should exponentially decay 1 quickly with n.

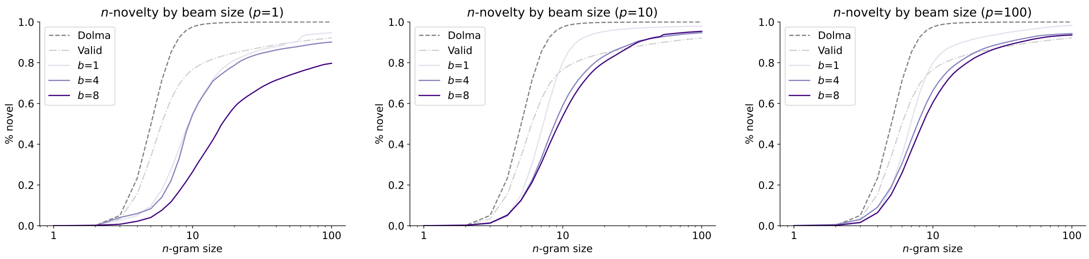
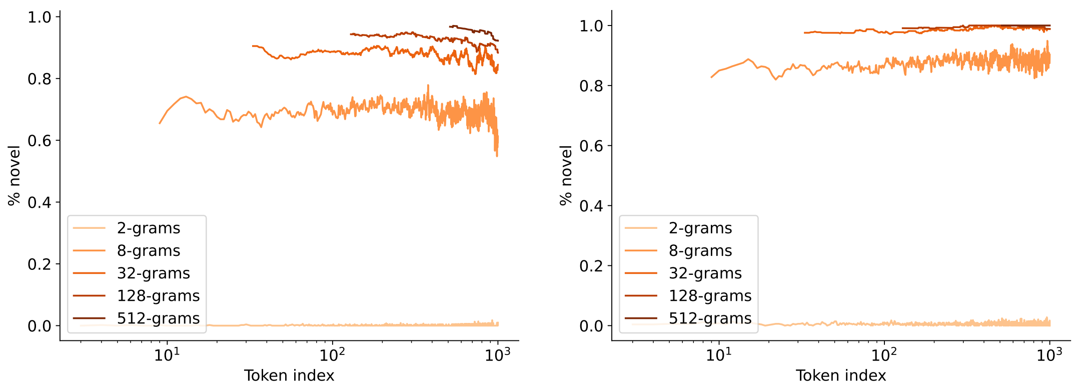


Figure 6: Beam decoding results with different amounts of training tokens used as a prompt: 1 token (left), 10 tokens (center), and 100 tokens (right).



(a) Validation

(b) Pythia-12B

Figure 7: n-novelty by token index in validation and Pythia-generated text.

C.2 Probably Nondeterministic Case

The previous derivation assumes that all tokens are nondeterministic. We relax this slightly by assuming that, within an n-gram, some tokens can be deterministic, but some fixed rate r will not be. More formally, we assume there exists r such that, for all x, $p(X_i = x_i \mid X_{<i} = x_{<i}) \leq q$ with probability r (over i). This implies that the novelty is as follows:

$$\begin{aligned} p(X\ n\text{−novel}) &\geq 1 - \sum_{d \in \mathcal{D}_n} \prod_{i=1}^{n} p(X_i = d_i \mid X_{<i}) \\ &\geq 1 - \sum_{d \in \mathcal{D}_n} q^{rn} \cdot 1^{(1-r)n} \\ &= 1 - \sum_{d \in \mathcal{D}_n} q^{rn} \\ &= 1 - |C| \exp(n(\log r - \ell)). \end{aligned}$$

This is the form of the lower bound invoked in the main plots (cf. Section 4.3). We believe this assumption is possibly strong, but it allows us to get a first-pass lower bound.

D Beam Search Results Elaboration

The beam search decoding used in Figure 4 is deterministic because the temperature is 0 and the prompt is null. To complement these results, we also include additional results in Figure 6 where a prompt of length p (taken from the training data) is used. In this regime, we find that, similar to the promptless results, beam search decreases n-novelty. However, the novelty curve is not so extreme for beam size 8. This indicates that, with beam size 8, the LM does not always copy very large chunks of training documents like in Figure 4.

Novelty by Token Index

We also present an additional analysis showing how novelty varies across token index in the document, for both validation and LM-generated text. As shown in Figure 7, n-novelty stays roughly constant across different token positions across both validation and LM-generated text, for every value of n. This pattern in the validation text in particular (Figure 7a) suggests token index does not correlate with n-gram size, and is thus likely not a confounder in Section 6.