PREQUENTIAL PRETRAINING: LEARNING TO CONTINUALLY COMPRESS THE FUTURE

Anonymous authors
Paper under double-blind review

ABSTRACT

A major approach within computational theory of intelligence is prequential coding, which views continual compression of a stream of data as identifying parsimonious models for the underlying data. Prequential coding has been an important motivation for pretraining, leading to language models that can accurately complete single documents. Despite these successes, there has been little work on studying prequential coding in the more general setting of *temporal streams* of text data. Success in this setting would enable models to make predictions of future texts, across documents and potentially serve as an important primitive for continual learning. We study the empirical feasibility of this problem and curate a high-resolution, verifiably timestamped corpus with a single temporal stream of text up to 10B tokens spanning 2010-2022, including revisions to the same documents over time. We evaluate standard language modeling approaches as coding algorithms and find significant room for improvement in exploiting temporal and long-range structure. Natural interventions, such as long context, reach only 0.69 bits per byte (BPB), compared to retrieving exact-match prefixes from earlier in the stream which reaches 0.49 BPB, revealing substantial headroom for long-context models. Finally, we identify that in principle, it is possible to attain far better compression rates of 0.43 BPB through masking tokens that are in principle knowable from the past.

1 INTRODUCTION

Language modeling pretraining has been strikingly successful, and one major argument for its success has been the theory of prequential coding and Solomonoff induction (Dawid, 1984; Solomonoff, 1964; Delétang et al., 2024). In this view, a model which can perform efficient, online compression of a single stream of data will capture some aspects of the data-generating mechanism behind this stream (Rissanen, 1978; Solomonoff, 1978). In language modeling, this is often viewed as an argument that a sufficiently low perplexity model enables accurate continuations of arbitrary documents (Radford et al., 2019; Brown et al., 2020).

Following the empirical successes of pretraining on unordered documents, we take this idea one step further, and ask whether we can perform prequential coding of a temporal stream of naturally occurring text. In this case, the corresponding Solomonoff induction argument would say that a model with sufficiently low (cumulative) cross entropy would be able to accurately predict *future* texts from this stream.

Despite the importance of prequential coding of temporal data, and the growing interest in continual learning, surprisingly little is known about whether standard language modeling methods are capable of performing effective prequential compression, and what perplexities may be attainable. We define this task, construct a carefully curated dataset and evaluation, and adapt existing long-context training methods for this setting.

Although many works have studied temporal knowledge acquisition in language models, few works have studied this in the pretraining setting, and none have collected a verifiably timestamped, one-second granularity pretraining corpus. We build this corpus through a combination of Wikipedia, Stack Exchange, arXiv and Global Voices, totaling 13 years from 2010 until 2022 and 10 billion tokens, including revisions to the same articles, often millions of tokens apart. We treat this as

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.

a single, 10 billion token temporally ordered string, and ask whether existing or new pretraining methods achieve low prequential loss in this setting.

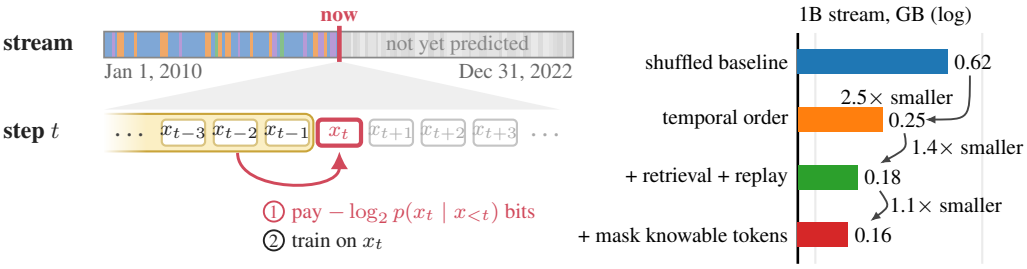


Figure 1: Prequential pretraining operates on a single temporal stream of text constructed from verifiably timestamped data such as Wikipedia. Models are trained online over time, and compared based on their cumulative, prequential loss. We find that existing models can extract bits from temporal structure, but there is significant headroom for improvement in long-context capabilities.

We first compare to existing approaches, including shuffled, i.i.d. training and long-context models, and find that significant gaps remain in modeling long, temporal streams of text. Through carefully defining our data format, we can perform apples-to-apples comparisons between training on temporal and shuffled streams, allowing us to measure how much the model benefits from temporal order. We find that the use of temporal information decreases prequential loss by 60% relative to shuffling documents, suggesting that models are able to exploit some temporal structure, but we also find that many of these benefits come from temporal locality rather than long-range structure, with little benefit beyond 32k sequence length.

To understand whether these losses are optimal, we study interventions adapted to the prequential setting, including replay and retrieval, and find significant gains are possible. Since we only incur prequential loss the first time we see a token, we are free to replay past data, with gains growing as the replay budget increases. Because much of the long-range structure lies far beyond the 32k context window, we mix in the continuation of the longest exact match from the past, yielding even larger gains. By combining retrieval with replay over all past data, we reduce loss by 30% on the 1B-token stream.

Finally, we show that there is even more headroom remaining for future models that better exploit long-range structure, by identifying tokens that are deterministically knowable given the past by masking the losses on those tokens. For example, in a standard diff, the text for a deleted line is fully deterministic once the model knows a specific line is being deleted, so a model with perfect recall would incur no loss. Masking these tokens shows that the loss of our best model can be reduced further by at least 12%, showing that prequential pretraining is a challenging and interesting task for long-context and pretraining research.

2 The Prequential Setting

The standard language modeling task consists of a generative modeling task over an independent and identically distributed collection of sequences $x_1..x_n \sim P \subseteq \mathcal{X}$. Usually, this is done by minimizing cross-entropy for some parametric family of models $f \in \mathcal{A}$ parametrizing a probability distribution over $\mathcal{X}$

$$\mathcal{L}_n = \sum_{i=1}^{n} -\log_2 P_f(x_i \mid x_1, \ldots, x_{i-1}), \quad f \in \mathcal{A} \tag{1}$$

This has been a surprisingly successful paradigm in enabling a range of downstream AI applications through subsequent post-training (Ouyang et al., 2022; Wei et al., 2022). One explanation for this success has been the theory of *prequential coding*, which argues that minimizing the overall code length of an online compression task is a promising path to building intelligent systems (Dawid, 1984; Delétang et al., 2024).

We formalize the *prequential* equivalent to the standard language modeling task in the following way. For a stream of bytes $x_1, \ldots, x_n \sim P$ drawn from a potentially stochastic generative process P, let $\mathcal{A}_k := \mathcal{A}(x_1 \ldots x_k)$ be a learning algorithm operating on the first k sequences/tokens/bytes and let $f_k \sim \mathcal{A}_k$ be one draw from $\mathcal{A}_k$. Then, the expected cumulative prequential code length in bits is

$$\mathcal{L}_n = \mathbb{E}_f \left[\sum_{i=1}^{n} -\log_2 P_{f_{i-1}}(x_i \mid x_1, \ldots, x_{i-1}) \right]. \tag{2}$$

Notice that the prequential formulation of the task and standard language modeling are quite similar. Both can be viewed as a log-likelihood over the entire stream $x_1 \ldots x_n$ but the standard language modeling setting evaluates a single model f (potentially fitted to the entire distribution) while the prequential setting is inherently online, with f_{i-1} conditional only on the past.

The online, prequential view opens up language modeling to a broader set of data generating models where the stream of text comes from non-i.i.d and non-stationary data sources. In this work, we consider *temporal* prequential streams, where the sequence of bytes $x_1 \ldots x_n$ comes from verifiably timestamped text sources. This view corresponds to treating the pretraining corpus as a single, long stream of text, where the model is now forced to not only compress a single document, but also be able to predict future documents.

2.1 Measuring the ability to use temporal information

Can we quantify the value of this temporal structure? Intuitively, there is a temporal causality to the world and we might expect there to be parsimonious models of the data in temporal order, much in the same way that research has found that models of text in left-to-right order often attain substantially lower perplexity than other factorizations (Papadopoulos et al., 2024).

To quantify this "value of time" in the prequential setting, we note that the core loss we consider in Equation 2 is the chain rule, and this chain rule expands to a log likelihood on the same joint distribution regardless of the factorization order. Because of this, we can easily compare prequential losses obtained by different factorizations to quantify the value of time in the amount of extra bits saved during compression.

Importantly, this chain-rule property also gives our task a type of representation invariance. We show that regardless of how we represent the stream of bits $x_1 \ldots x_n$ (e.g. as a stream of documents, or temporal diffs over documents) as long as our transformations are injective and the decoder is computable, we can directly compare total bits across data formats. We discuss these details in further detail as we define the precise textual representation of our data stream in Section 3.

3 A Temporal Corpus for Prequential Pretraining

Before we can study the effectiveness of prequential pretraining, we must first build a pretraining corpus which is represented as a single stream of text data, with a verified temporal ordering among each document. Despite the growing interest in continual learning and temporal knowledge acquisition for language models, much of the existing data relies on coarse grained timestamps (often at the year level) and does not capture the full richness inherent to a fully ordered temporal pretraining task (Jin et al., 2022; Pilchen et al., 2026; Jang et al., 2022).

Because of these issues, we built our own pretraining corpus based on publicly accessible and permissively licensed data, for which we have high confidence in both second-level timestamps and immutability (i.e. any timestamps cannot be backdated or modified).[1] Under these criteria, we filtered to four sources for our dataset: Wikipedia, Stack Exchange, arXiv and Global Voices (a newspaper), spanning the time period [Jan 1 2010 12am UTC, Jan 1 2023 12am), consisting of around 400 billion tokens, and including revisions to the same articles over time. See appendix A.2 for more details.

[1] Some filters, such as dropping reverted edits, use later information (Appendix A.2.5).

Each datum in the corpus is formatted along with metadata identifying the data source, time of data creation/modification, and a consecutive global id number. The global id allows us to shuffle the documents for our prequential compression comparisons while retaining the same underlying joint distribution, as the id number makes any shuffle fully invertible.

3.1 SUBSAMPLING THE CORPUS

For our studies, we do not want to constantly train on the full 400B token dataset due to the computational challenges of training prequential models on datasets of this size. In the standard language modeling setting, downscaling a dataset is (nearly) trivial, as we would simply subsample the dataset with or without replacement. However, the nonstationary and non-iid nature of our data poses significant challenges in downsampling.

When we downsample our corpus, we must take care to avoid leaking future information from the future. This prevents many intuitively appealing forms of subsampling, such as restricting Wikipedia to "interesting" articles such as articles about COVID-19 or related viral diseases. This type of selection would leak information to any prequential learner that any uninteresting articles that appear in the prequential stream are likely to become interesting in the future.

Similarly, we cannot do the standard naive subsampling of articles by taking a random set of articles from the current Wikipedia dump, as this sampling depends on the article frequency and distribution in 2026. To solve these issues, we use an online sampling scheme which selects any article by hashing the ids of each article and select the articles with hash bits below some threshold. We use this online sampler to make the 10B token stream and then downsample that subset of articles to even smaller subsets of articles to make the other datasets.

4 EVALUATING EXISTING PRETRAINING RECIPES

We now turn to our modeling questions for prequential pretraining. Can we take off-the-shelf pretraining approaches, apply them to the prequential setting, and obtain significant benefits from the temporal structure of our data?

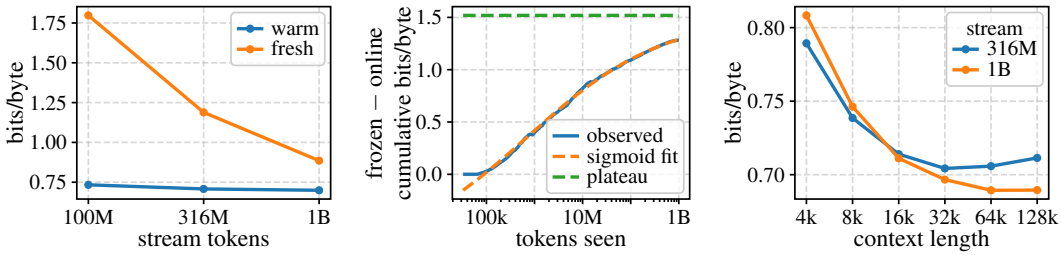
4.1 ADAPTING LANGUAGE MODELS TO THE PREQUENTIAL SETTING

Temporal, prequential pretraining and standard language modeling are quite similar, but there are important differences that we must handle when setting up our experiments. It is perhaps obvious that we cannot epoch or shuffle the data, but there are more subtle changes, such as the impact of batch size and context size on model performance.

Standard settings: Our training setting looks similar to most standard LLM pretraining and we use a 267M parameter dense transformer with the Qwen3 architecture (Yang et al., 2025a), with the exception of the use of multi-head attention. In later sections where we explore hybrid Gated DeltaNet architectures, we follow the Qwen3-Next family of architectures in design and hybrid ratio (Team, 2025). We use batch size 1, as larger batches make the model stale, degrading prequential loss (Section 4.2) and a 32k context length.

Learning rate schedules for prequential learning is a complicated topic. In theory, the online nature of prequential learning suggests that the learning rate schedule should be fully horizon-free (i.e. independent of the training horizon) and leverage tricks such as EMA of the training weights to obtain annealing effects. However, we found that all horizon free learning rate schedules underperformed horizon-dependent schedules in our setting for the prequential loss, and we opt to use a WSD learning rate (Hu et al., 2024) throughout our work (Appendix A.4).

Finally, we found in our experiments that our smaller token streams incurred high losses and high experiment variance due to the initial phase of training for language models requiring a large number of tokens before they can capture basic structures of the English language and our token stream. Because of this, we choose to warm-start our models using a model *pre-pretrained* on 5 billion tokens that existed prior to our start date of 2010, trained with standard, shuffled data. This substantially improves the small-scale experiments while having a smaller impact on our larger scale runs as seen in Figure 2a.

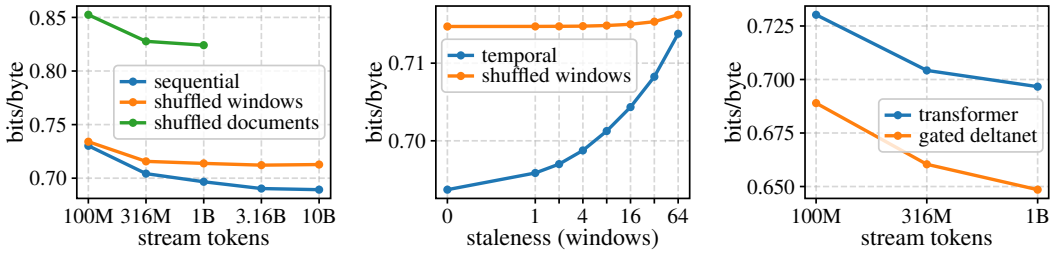


(a) Pre-pretraining improves models and stabilizes training

(b) Prequential training has predictable data scaling

(c) Prequentially pretrained models improve with longer context

Figure 2: We find that applying standard language modeling techniques to prequential pretraining provides familiar behaviors from the models, including stability from pretraining, data scaling, and gains from increasing context.



(a) Gains from models' ability to exploit temporal structure

(b) Temporal gains depend on frequent model updates

(c) Gated DeltaNet provides perplexity gains

Figure 3: Standard language model pretraining recipes such as long-context training provide gains in the prequential setting, giving large gains in bits/byte over shuffled training. However, these gains depend on computationally costly frequent model updates.

4.2 Existing pretraining recipes underutilize temporal structure.

We train prequential pretraining models on 316M and 1B token streams, studying the effectiveness of standard training recipes in this setting. We find some positive results in this setting with standard recipes, such as data scaling and improvements from temporal information. However, the gains from temporal information saturate quickly, with even large context failing to give continued improvements in bits-per-byte.

Data scaling. We first begin with an analysis of data scaling. As we train on increasing amounts of data, we expect improvements to model performance (Kaplan et al., 2020). Prequential training curves do not obviously show scaling laws, due to the nonstationary nature of the data stream as seen in Figure 10. However, we find that the difference between the online trained model and a baseline model that is frozen at the pre-pretrained state follows a highly smooth and predictable sigmoidal structure, giving us reasonable confidence that the prequential setting behaves like standard pretraining with respect to data scale.

Exploiting temporal structure. We also find that a straightforward online learning recipe which does continual pretraining on the data stream benefits from temporal information. We compare the model's performance under temporal and shuffled streams, taking the difference between the two as the model's ability to extract temporal information. We keep the global document id the same while shuffling, which makes this transformation invertible, and thus the prequential losses are directly comparable (Section 2).

The most straightforward approach of shuffling the documents across time, then applying diffs leads to extreme degradation in performance due to the large sizes of the resulting diffs +147% bits (Figure 12a). A more favorable shuffling comparison is to keep the same diff structure, but to shuffle the diffs across time. Even in this case, we find a significant +18% bits increase relative to temporal

training. Finally, as an additional comparison, we shuffle at the context window, rather than document, level. This is not strictly invertible and so not directly comparable, but this most favorable shuffle still results in +2.9% more bits, suggesting that gains from the use of temporal information are durable.

These temporal gains require careful treatment of the data, and aggressive computational optimization such as batching can lead to significant performance degradation. When we batch our data in K sized batches, some of the data in the batch is evaluated on a "stale" model, as we effectively only update our model every K examples. We isolate the cost of this delay by scoring windows with stale weights while keeping the training trajectory fixed (Appendix A.7). In Figure 3b we see that the prequential loss of the sequentially trained model worsens quickly as the model is more stale, while the randomly shuffled models retain most of their performance. Since stale models do much worse for sequential order, this leads us to use small or single batch gradient descent, relying on large context windows and context parallelism to improve GPU utilization.

Long-context models. The promising signs in exploiting temporal structure lead us to consider significantly longer context models, as basic analyses of the temporal data stream show that the dependencies across articles in the diffs can span millions of tokens. We test context lengths up to 128k tokens in this setting (Figure 2c), but find that improvements from long context seem to rapidly plateau at around 32k tokens. This point does shift to the right as we increase the total number of tokens in the data stream, but these gains become increasingly computationally expensive to obtain.

To address some of these computational complexities and further increase context length, we study hybrid Gated DeltaNet models, with the attention to GDN hybrid ratio set according to the Qwen3-Next models. We found that hybrid GDN models give significant improvement to bits per byte across various compute scales. While several of our runs relied on un-optimized GDN implementations, more optimized wallclock comparisons of the hybrid GDN architecture showed 1.1x to 2.2x speedups for 32k and 128k context respectively.

As GDN models are state space models, there is also a possibility of using them for *infinite* context extension, by training them using truncated backpropagation through time. In this case, we simply carry the GDN state across training batches, but do not propagate gradients back through this boundary. While in theory this allows for propagation of information across the entire pretraining corpus, in practice we find little gains from this approach of between 0.05% and 0.30% (Figure 14a) and diminishes as we increase the context length (Figure 14b).

5 Improved Models for Prequential Pretraining

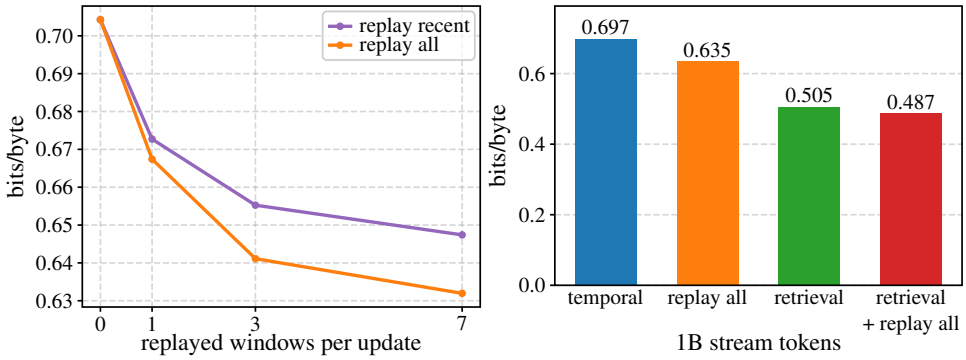
We find that existing pretraining recipes can exploit some temporal structure, but have some obvious limitations in context length and continual adaptation. We now consider new and classic techniques specific to prequential training to push the limits of performance for our current models.

Our approach to improving long-context behaviors centers on two differing lines of thought. In the first approach, we try to improve long-term memory abilities of the model through replay and retraining of past data. In the second approach, we consider more direct recall abilities by leveraging simple retrieval heuristics which bring distant, past data into more immediate context.

5.1 Replay

We have seen that models do not sufficiently leverage long-context information via attention or Gated DeltaNet, and most of the models' adaptation occurs *in-weights* via gradient descent. Past works on prequential compression (Bornschein et al., 2023) have made similar observations, and have argued that these capabilities can be amplified through additional training passes, where past data is retrieved and re-trained on between each step of prequential training.

The key to replay is what we retrieve and how to balance new vs old data. We first consider two simple approaches to retrieval. Retrieving among *all* past examples (`replay all`) and the most recent 512 context windows (`replay recent`). We initially hypothesized that much of the short range dependence in our dataset is short range, and thus short history replay could be beneficial. However, we found that replay of random samples from the full history significantly outperformed this approach (Figure 4b).



(a) Replay improves prequential loss, with gains from longer horizons.

(b) Retrieval composes with replay and provides large gains.

Figure 4: Techniques that are adapted to our prequential, single-stream setting provide further improvements with both replay and retrieval providing major gains.

Furthermore, we find that there is relatively strong scaling in performance as a function of replay rate, with more historical data replay per prequential update providing further gains (Figure 4b). One explanation for these results is that prequential pretraining is a highly data constrained setting, where it is quite important to fully learn details from the past to predict the future. In this view, replay serves as a form of epoching over data that improves data efficiency (Muennighoff et al., 2025).

5.2 Retrieval

The other approach we consider is retrieval of texts from the distant past. Our dataset contains high repetition where many Wikipedia articles follow the same structure, such as grammar fixes. Similarly, two versions of the same abstract often share the same ideas and two news articles about the same event spaced a week apart will share similarities. These repetitions are highly useful for modeling, and extremely long distance, with more than half of Wikipedia revisions spaced more than 32k tokens apart even in our 100M dataset. Despite this, we found in Figure 2c that our model saturates at context length 32k, which fails to use this rich source of information.

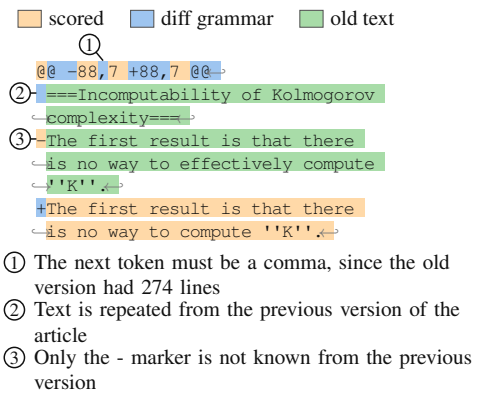
We take inspiration from the success of retrieval based methods such as knn-lm and infini-gram (Khandelwal et al., 2020; Liu et al., 2024) and build a retrieval-based prequential pretraining model. Since we particularly care about loss on repeated text, we do this by building a cache for exact match sequences. For each token, we find the longest matching sequence in our already-seen data (of at least 16 and at most 512 tokens) and add a learnable gate that reads the input to the LM head and mixes the retrieved continuation with our model's prediction (Appendix A.8). We find that retrieval leads to large gains of 27.6% (Figure 4b).

6 Headroom Analysis

Finally, we probe the theoretical limits of prequential pretraining. Is it possible that our current architectures and training recipes get close to the best possible learning rates in this setting, or do we have significantly more headroom?

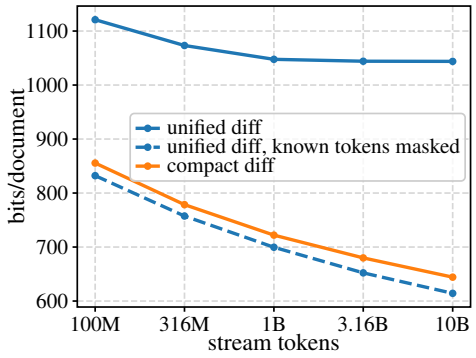
Recall that any *invertible* data format produces a comparable and valid prequential loss. This leads us to optimize our data format over the space of formats, finding human-uninterpretable but highly nonredundant formats that significantly reduce the prequential loss.

Our main observation is that the standard diff format is highly human interpretable, but has significant redundancy, as these diffs often contain significant amounts of repeated content. Not only does this require the model to perfectly remember what the state of the Wikipedia article was at the previous timestamp, but it also needs to surgically update its own state of what the article looks like.



① The next token must be a comma, since the old version had 274 lines
② Text is repeated from the previous version of the article
③ Only the - marker is not known from the previous version

(a) Fully determined tokens in a unified diff



(b) More compact diff formats and masking fully determined tokens provide additional major gains

Figure 5: We identify significant further gains are possible by showing that extreme compression of the diffs and loss masking on fully knowable tokens lead to major gains.

This is a challenging problem for neural models updated via gradient descent, and we ask whether we can find an alternative that removes these redundancies.

6.1 Compact Diffs and Full Text.

We design a diff that has minimal redundancy by encoding each revision with the fewest bytes from a small set of invertible operations. For instance, inserting an "s" after the fifth character becomes `+5:s`, and changing "a shortest" to "the shortest" within a sentence becomes `*1Q,1:the`, which is 9 characters, much less than unified diff that must repeat the full sentence twice. We provide a full construction and an argument for invertibility of this transform in Appendix A.3.2.

With this minimal diff format, prequential training attains significantly lower bits per document, despite being a lossless transformation of the same underlying stream of text. Bits per document drops from around with the original diff format attaining 1048 bits while the new compact diff further pushes this to 722.

As a control, we also study a transformation in the opposite transformation, maximizing redundancy by repeating the full article texts every time that there is a change. Information theoretically, this is the same task as modeling the git diffs, but we find that bits per document increases rapidly, with the 1B token case having double the compact diff of 1564 bits per document.

These analyses suggest that these temporal prequential models still lack the ability to model simple long-range structure, making this dataset a useful challenge for long-context models with significant amounts of headroom.

6.2 Masking Knowable Tokens

The significant gains of the compact diff might be due to many reasons, such as the shorter overall length of the sequence, or the reduced redundancy. To disambiguate some of these causes, we study a second intervention which keeps the original diff format, but identifies segments such as deletions which are perfectly knowable given the past documents, and masks those losses.

More specifically, a unified diff contains many "perfectly knowable" tokens (Figure 5a). If we are told the line number for a document, and we know that the operation in the diff is deletion, we can look up the original document and do an exact recall of the original line. A stronger model which knows the rules of the diff format and has perfect recall can therefore avoid incurring any losses on these positions.

We consider the loss constructed by masking such knowable tokens, and evaluate the model in this setting. In this case, we find that this masked loss achieves very similar but slightly lower losses

than the compact diff (Figure 5b), suggesting that the key factor in the loss improvement is due to the removal of redundant context tokens. Additionally, we do find nontrivial gains in this format rather than the compact diff, suggesting that the complex format of the compact diff format may pose challenges to prequential learning at our compute scales.

The combination of these gains suggests that there is major room for improvement in existing algorithmic approaches to prequential pretraining and our ability to fully extract information from the temporal structure in the world.

7 Related Work

Prequential coding. Strong predictions give short codes, making compression a useful lens for studying language models (Delétang et al., 2024; Heurtel-Depeiges et al., 2025). Prequential coding takes an online learning view (Dawid, 1984), enabling studies of neural description lengths (Blier & Ollivier, 2018; Bornschein et al., 2023). The resulting learning curves also inform estimates of learnable structure in work on epiplexity (Finzi et al., 2026). Our work builds upon these ideas, and advances an empirical study of *temporal* prequential coding for language model pretraining.

Online and continual learning. Continual learning (Parisi et al., 2019) seeks to acquire new information without catastrophically forgetting past knowledge (McCloskey & Cohen, 1989). For language models, this motivates lifelong pretraining (Jin et al., 2022) and methods based on replay and targeted parameter updates (Rolnick et al., 2019; Fernandez et al., 2024). The cost of retraining further motivates scalable recipes for reusing checkpoints through learning-rate scheduling and data mixing (Gupta et al., 2023; Ibrahim et al., 2024; Parmar et al., 2024) as well as recurrent models (Marschall et al., 2020). Our work is closest to those on dynamic evaluation, which update models on observed text at test time, (Krause et al., 2018; Rannen-Triki et al., 2024) but differ in our focus on a single, online corpus rather than test-time adaptation.

Temporal datasets and benchmarks. Language models become less accurate on text from beyond their training period (Lazaridou et al., 2021), motivating works that study temporal knowledge acquisition (Jang et al., 2022; Li et al., 2025). Works closest to ours study chronologically ordered pretraining (Pilchen et al., 2026), but differ in underlying motivation. Prior works focus on temporal knowledge acquisition, and thus use coarse grained, year level data with weak provenance (e.g. common crawl dumps) which are sufficient for their purposes, but insufficient for constructing a single, temporally ordered text stream necessary for prequential coding. Additionally, our goal of prequential compression leads to different objectives (cumulative loss) that allow us to precisely quantify the value of temporal information, and force us to incorporate continual learning methods.

Long context and retrieval. The growing cost of full attention motivates compressing context into recurrent states, as in Gated DeltaNet (Yang et al., 2025b), or adaptive weights through test-time training (Sun et al., 2025; Tandon et al., 2025). Retrieval complements such compressed memories with explicit access to past text, improving predictions through retrieved continuations or additional training (Khandelwal et al., 2020; Liu et al., 2024; Hardt & Sun, 2024). Our work leverages these advances, and find that while they improve prequential compression, significant gaps remain in modeling.

8 Discussion

The success of classical ideas from online learning, such as replay and retrieval, suggests that standard pretraining recipes leave much of the temporal structure in the stream unused, encouraging us to revisit how language models store and recall the distant past. Longer context gives little benefit beyond 32k tokens and carrying recurrent state across the entire stream helps by less than 0.3%, yet revisions to the same article are often millions of tokens apart, and masking tokens that are fully determined by the past shows that the loss of our best model can be reduced by at least a further 12.1%. We are excited by methods that close this gap, such as memory that learns what to retrieve from the past and what to write into weights, and by horizon-free optimizers and systems that make frequent, small-batch updates efficient, since these are central to continual learning as well as to prequential pretraining. We hope that measuring cumulative code length on a single temporal stream inspires models that make better use of the past to predict the future.

AI USE STATEMENT

In this work, we used generative AI tools to propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, clean and reformat the dataset, support qualitative and thematic data analysis, interpret results, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, and generating synthetic data sets and assist with translation are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images, create or edit software code, creation of artifacts, summarize or analyse existing literature, discover research topics or identify gaps, brainstorming, sourcing/searching for information, edit a research paper to improve readability, identify relevant literature. We also used AI to write parts of the abstract. We have reviewed all AI-assisted work. We skimmed all AI-generated code and read closely and hand wrote the most important pieces, such as the data processing pipeline. Model code and evaluation were carefully compared against other codebases. We have read over all AI writing to make sure of it is correct. We carefully reviewed and understood any mathematical proofs made by AI. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We hope that our work leads to language models that make better use of the past to predict the future, with applications to continual learning. Our temporal corpus is built from public, openly licensed sources. The warm-start corpus uses LDC and Institutional Books data under their research licenses, which we do not redistribute. We believe most other harms specific to our work apply to general language modeling research.

REPRODUCIBILITY STATEMENT

We will open-source our code for building the temporal corpus, training and evaluation. We will make the temporal corpus available for download.

REFERENCES

Léonard Blier and Yann Ollivier. The Description Length of Deep Learning models. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Jorg Bornschein, Yazhe Li, and Marcus Hutter. Sequential learning of neural networks for prequential MDL. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=dMMPUvNSYJr.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

A. P. 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):278–292, 1984. ISSN 0035-9238. doi: 10.2307/2981683. URL https://www.jstor.org/stable/2981683.

Aaron Defazio. ScheduleFree+: Scaling Learning-Rate-Free & Schedule- Free Learning to Large Language Models, May 2026. URL http://arxiv.org/abs/2605.19095. arXiv:2605.19095 [cs.LG] version: 1.

Aaron Defazio, Xingyu Alice Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. The road less scheduled, 2024. URL https://arxiv.org/abs/2405.15682.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. Language Modeling Is Compression, March 2024. URL http://arxiv.org/abs/2309.10668. arXiv:2309.10668 [cs.LG].

Jared Fernandez, Yonatan Bisk, and Emma Strubell. Gradient Localization Improves Lifelong Pretraining of Language Models, November 2024. URL http://arxiv.org/abs/2411.04448. arXiv:2411.04448 [cs.CL] version: 1.

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, 2026. URL https://arxiv.org/abs/2601.03220.

Kshitij Gupta, Benjamin Thérien, Adam Ibrahim, Mats L. Richter, Quentin Anthony, Eugene Belilovsky, Irina Rish, and Timothée Lesort. Continual pre-training of large language models: How to (re)warm your model?, 2023. URL https://arxiv.org/abs/2308.04014.

Moritz Hardt and Yu Sun. Test-Time Training on Nearest Neighbors for Large Language Models, February 2024. URL http://arxiv.org/abs/2305.18466. arXiv:2305.18466 [cs.CL].

David Heurtel-Depeiges, Anian Ruoss, Joel Veness, and Tim Genewein. Compression via Pretrained Transformers: A Study on Byte -Level Multimodal Data, May 2025. URL http://arxiv.org/abs/2410.05078. arXiv:2410.05078 [cs.LG].

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, Xinrong Zhang, Zheng Leng Thai, Kaihuo Zhang, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. Minicpm: Unveiling the potential of small language models with scalable training strategies, 2024. URL https://arxiv.org/abs/2404.06395.

Adam Ibrahim, Benjamin Thérien, Kshitij Gupta, Mats L. Richter, Quentin Anthony, Timothée Lesort, Eugene Belilovsky, and Irina Rish. Simple and scalable strategies to continually pre-train large language models, 2024. URL https://arxiv.org/abs/2403.08763.

Joel Jang, Seonghyeon Ye, Changho Lee, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, and Minjoon Seo. TemporalWiki: A Lifelong Benchmark for Training and Evaluating Ever-Evolving Language Models. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 6237–6250, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.418. URL https://aclanthology.org/2022.emnlp-main.418/.

Xisen Jin, Dejiao Zhang, Henghui Zhu, Wei Xiao, Shang-Wen Li, Xiaokai Wei, Andrew Arnold, and Xiang Ren. Lifelong pretraining: Continually adapting language models to emerging corpora, 2022. URL https://arxiv.org/abs/2110.08534.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through Memorization: Nearest Neighbor Language Models, February 2020. URL http://arxiv.org/abs/1911.00172. arXiv:1911.00172 [cs.CL].

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. Dynamic evaluation of neural sequence models. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2766–2775. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/krause18a.html.

Angeliki Lazaridou, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Tomas Kocisky, Sebastian Ruder, Dani Yogatama, Kris Cao, Susannah Young, and Phil Blunsom. Mind the Gap: Assessing Temporal Generalization in Neural Language Models, October 2021. URL http://arxiv.org/abs/2102.01951. arXiv:2102.01951 [cs.CL].

Jeffrey Li, Mohammadreza Armandpour, Iman Mirzadeh, Sachin Mehta, Vaishaal Shankar, Raviteja Vemulapalli, Samy Bengio, Oncel Tuzel, Mehrdad Farajtabar, Hadi Pouransari, and Fartash Faghri. TiC-LM: A Web-Scale Benchmark for Time-Continual LLM Pretraining, June 2025. URL http://arxiv.org/abs/2504.02107. arXiv:2504.02107 [cs.LG].

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

Owen Marschall, Kyunghyun Cho, and Cristina Savin. A unified framework of online learning algorithms for training recurrent neural networks. *Journal of Machine Learning Research*, 21 (135):1–34, 2020. URL http://jmlr.org/papers/v21/19-562.html.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An Empirical Model of Large-Batch Training, December 2018. URL http://arxiv.org/abs/1812.06162. arXiv:1812.06162 [cs.LG].

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In Gordon H. Bower (ed.), *Psychology of Learning and Motivation*, volume 24 of *Psychology of Learning and Motivation*, pp. 109–165. Academic Press, 1989. doi: https://doi.org/10.1016/S0079-7421(08)60536-8. URL https://www.sciencedirect.com/science/article/pii/S0079742108605368.

Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models, 2025. URL https://arxiv.org/abs/2305.16264.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

Vassilis Papadopoulos, Jérémie Wenger, and Clément Hongler. Arrows of time for large language models. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 39509–39528. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/papadopoulos24a.html.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2019.01.012. URL https://www.sciencedirect.com/science/article/pii/S0893608019300231.

Jupinder Parmar, Sanjev Satheesh, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Reuse, don't retrain: A recipe for continued pretraining of language models, 2024. URL https://arxiv.org/abs/2407.07263.

Hippolyte Pilchen, Romain Fabre, Franck Signe Talla, Patrick Perez, and Edouard Grave. Understanding Data Temporality Impact on Large Language Models Pre-training, May 2026. URL http://arxiv.org/abs/2605.22769. arXiv:2605.22769 [cs.CL].

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

Amal Rannen-Triki, Jorg Bornschein, Razvan Pascanu, Marcus Hutter, Andras György, Alexandre Galashov, Yee Whye Teh, and Michalis K. Titsias. Revisiting Dynamic Evaluation: Online Adaptation for Large Language Models, March 2024. URL http://arxiv.org/abs/2403.01518. arXiv:2403.01518 [cs.CL].

J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978. ISSN 0005-1098. doi: https://doi.org/10.1016/0005-1098(78)90005-5. URL https://www.sciencedirect.com/science/article/pii/0005109878900055.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience Replay for Continual Learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://papers.neurips.cc/paper_files/paper/2019/hash/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Abstract.html.

Ray Solomonoff. Complexity-based induction systems: comparisons and convergence theorems. *IEEE transactions on Information Theory*, 24(4):422–432, 1978.

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

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (Learn at test time): RNNs with expressive hidden states. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57503–57522. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/sun25h.html.

Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, Carlos Guestrin, Jed McCaleb, Yejin Choi, and Yu Sun. End-to-End Test-Time Training for Long Context, December 2025. URL http://arxiv.org/abs/2512.23675. arXiv:2512.23675 [cs.LG].

Qwen Team. Qwen3-next: Towards ultimate training & inference efficiency. *See qwen. ai/blog*, 2025.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2022. URL https://arxiv.org/abs/2109.01652.

Kaiyue Wen, Zhiyuan Li, Jason Wang, David Hall, Percy Liang, and Tengyu Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective, 2024. URL https://arxiv.org/abs/2410.05192.

Ronald J. Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2(4):490–501, 1990.

Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025b. URL https://arxiv.org/abs/2412.06464.

A Appendix

A.1 Training recipe and model selection

All models in the paper follow a single recipe, which we detail here. Unless stated otherwise, every run uses these settings.

Architecture. Our default model is a dense Qwen3-style transformer (Yang et al., 2025a) with 267,949,056 parameters. It has 16 layers, hidden size 1024, 8 attention heads of dimension 128 with full multi-head attention (8 key-value heads), and SwiGLU MLPs of width 2752. It uses pre-norm RMSNorm with QK-norm, RoPE with $\theta = 10^6$, untied input and output embeddings, and the 32,000-token Llama 2 tokenizer. The hybrid model (280,874,688 parameters) keeps full attention in every fourth layer and replaces the other 12 layers with Gated DeltaNet (Yang et al., 2025b), following the 3:1 ratio of Qwen3-Next (Team, 2025). Each Gated DeltaNet layer has 8 heads of dimension 128 and a causal convolution of width 4. We compute in bf16 and keep fp32 master weights.

Prequential training. The stream is the concatenation of all documents in temporal order, each wrapped in BOS and EOS tokens. We cut it into non-overlapping windows of 32,768 tokens that ignore document boundaries, and train with batch size 1. Each window is scored by the weights from before the update on that window (test-then-train), so every token is predicted by a model that has seen only the past. No state is carried across windows, except in the infinite-context Gated DeltaNet runs. Bits per byte is the summed loss in bits divided by the UTF-8 bytes of the rendered stream, metadata headers included. For 64k and 128k contexts we extend RoPE with YaRN, scaled by the ratio of the new context length to 32k. The hybrid warm start additionally trains for 256 updates at the new length.

Optimization. We use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, weight decay 0.1 on all parameters, and gradient clipping at norm 1.0. The WSD schedule (Hu et al., 2024) warms up linearly for 13 updates and holds the peak for 313. It then decays with a cosine to $0.1\times$ the peak over the remaining updates, which cover 89% of the 100M stream and over 99% of the 10B stream. The online run loads only the weights of the warm start, and its optimizer state starts fresh.

Warm start. Every online run starts from a checkpoint pre-pretrained on 5B tokens of text written before 2010: arXiv, PubMed, Gigaword and New York Times articles, topped up with digitized books. These documents are shuffled. Pre-pretraining uses the same optimizer with batch size 8, 32k context and 19,073 updates, at a constant learning rate of 4×10^{-4} after 200 warmup updates. Each seed has its own warm start, so seeds differ in initialization but not in data order.

Replay and retrieval. At each update, replay samples three earlier windows uniformly without replacement, either from all past windows (`replay all`) or from the most recent 512 (`replay recent`). The update is the equal-weight average of the gradient on the current window and the mean gradient on the replayed windows, and replayed windows are never scored. The retrieval cache indexes each token of previous windows by a hash of its 16 preceding tokens and keeps the 4 most recent positions per hash. At every position, we extend the candidate matches backwards up to 512 tokens and keep the longest ones. The empirical distribution of the tokens that follow them gives the retrieval distribution p_{ret}. A gate MLP ($1024 \to 64 \to 1$) on the final hidden state outputs $\lambda = 0.95\,\sigma(\cdot)$, initialized to 0.05, and $\lambda = 0$ when there is no match. The model predicts $(1 - \lambda)\, p_{\text{LM}} + \lambda\, p_{\text{ret}}$. The whole network is trained on the loss of this mixture, with the gate using its own Adam optimizer at a constant learning rate of 10^{-3}.

Model selection. The only hyperparameter we tune is the peak learning rate. For the temporal and shuffled baselines, we sweep $\{0.6, 1.2, 2.4, 4.8, 9.6\} \times 10^{-4}$ at 100M and 316M, drop 0.6×10^{-4} at 1B, 3.16B and 10B, and report the peak with the best seed-mean final bits/byte. The selected peaks are:

- temporal: 4.8×10^{-4} up to 1B, and 2.4×10^{-4} at 3.16B and 10B;
- shuffled: 4.8×10^{-4} at 100M and 316M, 2.4×10^{-4} at 1B and 3.16B, and 1.2×10^{-4} at 10B.

Interventions use smaller sweeps centered on the temporal baseline's peak, and we again report the best peak per arm and size. Replay arms, alone and combined with retrieval, run at that peak

and one step either side ($\{2.4, 4.8, 9.6\} \times 10^{-4}$) at 100M, 316M and 1B. Retrieval alone gets the same three peaks at 1B. All other intervention runs use only the baseline's peak. Revision formats sweep $\{0.25, 0.5, 1, 2, 4\}\times$ the peak. The hybrid model is swept at 100M and 316M, and the 316M winner (4.8×10^{-4}) is reused at 1B and 3.16B. The replay and retrieval settings above come from preliminary experiments and are not tuned on the final streams.

Because the prequential loss is itself what we report, we select on it directly rather than on a held-out stream. In code-length terms this costs little: naming one of K configurations takes $\log_2 K$ bits, which is under 3 bits for our grids and negligible next to the more than 10^8 bits of even the smallest stream. Baselines, context-length, hybrid and shuffling results average two seeds, while replay, retrieval and diff-format results use one.

A.2 The temporal corpus

We describe how the stream of Section 3 is built, sampled and filtered, and which temporal guarantees hold afterwards. All sources are restricted to [2010-01-01, 2023-01-01) UTC.

A.2.1 Sources and timestamps

Wikipedia. English Wikipedia full-history dump of 2026-07-01, article namespace including redirects. A record is a revision, timestamped at save time (1 s resolution). Before windowing, we clean each page's full history: (i) if a revision's text equals an earlier revision's, we drop it and all revisions in between (reverts); (ii) we drop revisions whose diff against the previous surviving revision is empty or over 20k characters, or contains long punctuation runs, profanity (substring match), or more than 10 URLs, 150 template/citation markers or 50 table rows; (iii) we exclude the four "List of human protein-coding genes" pages. This keeps 644.6M of 884.0M revisions, with reverts accounting for 171.0M of the removals. Each kept revision is diffed against the previous *surviving* one, so every record is a real historical state of the page.

Stack Exchange. Data dump of 2026-06-30 (`stackexchange_20260630`), English-language sites. We rebuild every version of every question and answer from the post history, grouping entries by revision id and dropping versions that change none of title, body and tags. A record is a post version, timestamped at 1 ms resolution. It shows the thread title at that time, post type, tags, Markdown body, the user who made the version (so an edit's `Author` is its editor), and any edit summary.

arXiv. Version lists come from the Kaggle metadata snapshot (2026-08-22), and each version's title, authors and abstract (no full text) from the arXiv API. A record is a version, timestamped at submission. We drop papers with non-monotone or gapped version histories, and versions whose metadata equals the previous version's (e.g. PDF-only updates).

Global Voices. All published English posts from the WordPress API (104,322 posts, 2026-08-20), converted from HTML to text and timestamped at publication. Old versions are unavailable, so we keep only posts never modified after publication, which guarantees that both text and timestamp are as published.

Eligibility. arXiv papers and Stack Exchange threads must start inside the window. Wikipedia pages need one surviving in-window revision; pages created before 2010 enter in full at their first in-window revision (labeled `new-to-corpus` rather than `new`). Only in-window records are kept.

A.2.2 Ordering and rendering

Ordering. Records are sorted by $(t, h(u), r)$: timestamp, a 64-bit keyed BLAKE2b hash of the record's unit (§A.2.3), and a within-unit tie-breaker (revision id, post-history id or version number). Ties are therefore broken independently of content, and they are rare: 2,318 of 186,747 records in the 100M stream share a timestamp, and 4 share one across sources. A record's position is its global id.

Rendering. Figure 6 shows examples. Each record has a header with the global id, source, timestamp, title and, for Wikipedia, the event type. Wikipedia and Stack Exchange bodies are unified diffs against the previous version of the same object in the stream (first versions in full, as an all-added diff for Wikipedia); arXiv and Global Voices records are always in full. Every diff references an ear-

```
8 Wiki: 2010-01-01 21:48:59.000 | new |
Visszafolyó-patak

@@ -0,0 +1 @@
+#REDIRECT [[Visafoio Creek]]
\ No newline at end of file

13542 Wiki: 2011-07-21 19:44:35.000 |
revision | Visszafolyó-patak

@@ -1 +1 @@
-#REDIRECT [[Visafoio Creek]]
\ No newline at end of file
+#REDIRECT [[Mureş River]]
\ No newline at end of file

14757 Wiki: 2011-09-02 01:46:30.000 |
revision | Visszafolyó-patak

@@ -1 +1 @@
-#REDIRECT [[Mureş River]]
\ No newline at end of file
+#REDIRECT [[Mureş River]]
\ No newline at end of file

83052 Wiki: 2016-05-06 16:19:26.000 |
revision | Visszafolyó-patak

@@ -1 +1 @@
-#REDIRECT [[Mureş River]]
\ No newline at end of file
+#REDIRECT [[Mureş (river)]]
\ No newline at end of file

2115 Wiki: 2010-04-24 19:35:09.000 | revision
| Pope John XXIII

@@ -191,7 +191,6 @@
[[Category:Pope Paul VI]]
[[Category:Cancer deaths in Vatican City]]
[[Category:20th-century venerated
Christians]]
-[[Category:20th-century Italian people]]
[[Category:Burials at Saint Peter's Basilica,
Vatican]]
[[Category:Anglican saints]]
```

```
5300 Global Voices: 2010-09-25 09:00:27.000 |
China: Ai Weiwei's English Tweets

A group of volunteers help prominent
artist-activist, Ai Weiwei, to translate his
Chinese tweets into English.

73760 arXiv: 2015-10-23 13:52:01.000 |
Khovanov-Rozansky homology and Directed
Cycles

Authors: Hao Wu

We determine the cycle packing number of a
directed graph using elementary projective
algebraic geometry. Our idea is rooted in the
Khovanov-Rozansky theory. In fact, using the
Khovanov-Rozansky homology of a graph, we
also obtain algebraic methods of detecting
directed and undirected cycles containing a
particular vertex or edge.

2184 stackoverflow.com: 2010-04-28
13:49:02.203 | Post 2729998 Question 2729829

Title: Any way to profile code for cache
behavior? (Windows / C++)
Type: Answer
Author: M. Williams
Tags: --

Try `AQTime`. I'm pretty sure that some of it
options include cache profiling.

6868 stackoverflow.com: 2010-12-04
09:32:17.553 | Post 4352857
Edit summary: edited title

@@ -1,6 +1,6 @@
-Title: What is a coma separated set of
assignments?
+Title: What is a comma separated set of
assignments?
Type: Question
-Author: j riv
+Author: Sachin Shanbhag
Tags: c

I noticed in a routine
```

Figure 6: Records from the 100M stream (BOS/EOS omitted). Left: the full history of a Wikipedia redirect retargeted three times over six years (global ids 8 to 83,052), then an ordinary revision. Right: Global Voices, arXiv, a Stack Exchange answer, and a Stack Exchange edit whose `Author` is the editor.

lier record, so the stream is decodable online and the rendering is injective. Records are tokenized with the TinyLlama v1.1 tokenizer and wrapped in BOS/EOS, and BPB is computed over the UTF-8 bytes of the rendered records.

A.2.3 Sampling the stream

We sample *units* (Wikipedia pages, Stack Exchange threads, arXiv papers and Global Voices articles) and keep all their in-window records, so revision chains stay intact. For each source, we keep the K eligible units with the smallest $h(u)$, which is equivalent to a hash threshold. This depends only on identifiers, never on content or later popularity, and makes each stream a subset of every larger one. K grows by about $\sqrt{10}$ per size, which keeps the source mix fixed (Table 1).

A.2.4 Statistics

Wikipedia and Stack Exchange each provide about 48% of the tokens, arXiv 4.5% and Global Voices 0.07% (Table 1). Most of the stream is revisions: they hold 68.5% of the tokens in the 100M stream, and 97.3% of its Wikipedia tokens (Table 2).

Table 1: Stream sizes. Tokens include BOS/EOS; bytes are UTF-8 bytes of the rendered records.

	100M	316M	1B	3.16B	10B
Sampled units					
Wikipedia pages	1,719	6,044	19,984	62,764	196,388
Stack Exchange threads	29,413	92,342	293,660	927,990	2,937,242
arXiv papers	10,874	34,464	110,193	348,210	1,102,136
Global Voices articles	132	462	1,458	4,605	14,577
Tokens (millions)					
Wikipedia	47.72	150.25	477.09	1,512.35	4,760.02
Stack Exchange	47.65	150.75	480.49	1,516.44	4,801.17
arXiv	4.49	14.21	45.47	143.64	455.28
Global Voices	0.07	0.22	0.75	2.31	7.23
Total	99.93	315.44	1,003.80	3,174.75	10,023.69
Records	186,747	598,748	1,924,357	6,077,538	19,219,707
Bytes (GB)	0.29	0.91	2.89	9.15	28.82
Bytes per token	2.87	2.89	2.88	2.88	2.88

Table 2: The 100M stream by source. Objects are pages, posts, papers or articles; a revision is a record whose object appeared earlier.

	Units	Objects	Records	Revisions	Tokens (M)	Bytes (M)	Revision tokens
Wikipedia	1,719	1,719	46,446	44,727	47.72	129.31	97.3%
Stack Exchange	29,413	72,742	126,014	53,272	47.65	140.92	44.0%
arXiv	10,874	10,874	14,155	3,281	4.49	16.79	24.2%
Global Voices	132	132	132	0	0.07	0.25	0.0%
Total	42,138	85,467	186,747	101,280	99.93	287.26	68.5%

Revision distances. For each revision in the 100M stream, we measure the tokens and time since the end of the previous version of the same object (Figure 7, Table 3). About 30% of Wikipedia and Stack Exchange revisions directly follow their previous version, but the tail is long. Half of the Wikipedia revisions lie beyond a 32k context and 22% beyond 1M tokens; for arXiv, the figures are 97% and 75%. The maximum is 98.2M tokens, nearly the whole stream.

A.2.5 Retrospective selection and the temporal guarantee

Our snapshots are from mid-2026, and some of our filters look past a record's timestamp.

What holds. (i) Every record body is the exact text of its object at its timestamp, and every diff references an earlier record. (ii) Records are in timestamp order, with content-independent tie-breaking. (iii) Sampling depends only on identifier hashes.

What does not hold. Which records exist, and two header fields, depend on the state of the sources in 2026:

- *Wikipedia reverts.* A revision is dropped if it is ever reverted, possibly years later, and the revert is dropped with it. The stream is therefore the history of edits that survived, with little vandalism. The diff filters inherit this dependence because they compare against the previous surviving revision.
- *Global Voices.* Keeping only never-edited posts means that a post's presence reveals it was not edited later.
- *Stack Exchange.* Posts and threads deleted before 2026 are absent.
- *arXiv.* The version-history check also looks at versions submitted after 2022.
- *Names.* Wikipedia titles are the 2026 titles for all revisions, and Stack Exchange display names come from the 2026 user table.

All orderings and formats we compare contain the same versions, so their loss differences are unaffected. Absolute losses, however, measure how well the model predicts surviving content, which is likely easier than predicting everything that was posted; we have not measured by how much.

Table 3: Distance from each revision to the previous version of its object (100M stream).

	Revisions	Adjacent	Median (tokens)	90th pct. (tokens)	>32k	>1M	Median gap (time)
Wikipedia	44,727	29.7%	37,125	3.57M	50.8%	22.2%	2.1 days
Stack Exchange	53,272	31.6%	723	7.86M	21.6%	14.5%	44 min
arXiv	3,281	0.0%	2.93M	11.04M	97.3%	74.8%	131 days
All	101,280	29.7%	2,858	5.23M	36.9%	19.8%	3.2 hours

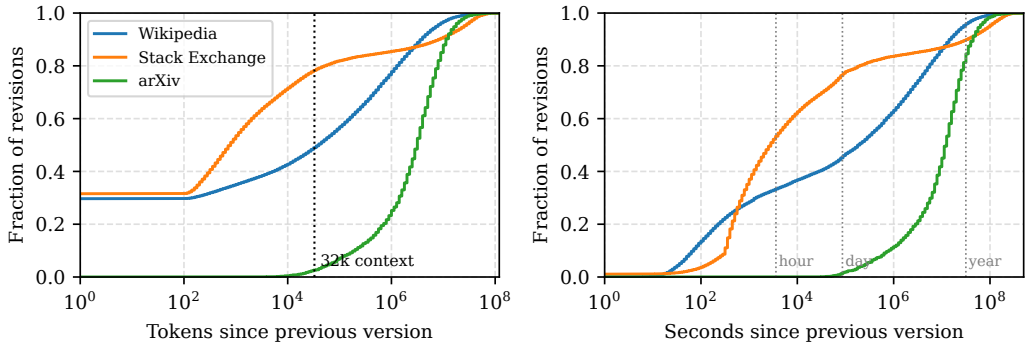


Figure 7: CDF of the distance from each revision to the previous version of its object (100M stream), in tokens (left; dotted: 32k context) and seconds (right).

A.2.6 Warm-start corpus

The warm-start corpus consists of 5B tokens of English text written before 2010: arXiv (Common Pile), English Gigaword (LDC2011T07, excluding NYT), the NYT Annotated Corpus (LDC2008T19) and PubMed (Common Pile), topped up with Institutional Books 1.0 and shuffled by document. Each document has a header in the stream's format, dated at the precision its source provides.

A.3 Lossless representations, compression accounting, and knowable-token masking

A.3.1 Encoded object and side information

At each stream size, the encoded object is the record sequence $D = (r_1, \ldots, r_N)$ of Appendix A.2.2. Each record holds a global id, source, timestamp, header fields and the full content of one version. A representation F maps D to the token stream $x^F = \bigoplus_i \texttt{BOS}\, \tau(R_F(r_i; r_{<i}))\, \texttt{EOS}$, where τ is the tokenizer. A revision's body may refer to the previous version of the same page (keyed by title) or post (keyed by site and post id), and that version always appears earlier in the stream. Headers are identical across formats. Formats differ only in the bodies of Wikipedia and Stack Exchange revisions, and orders only in the sequence of records.

A decoder reads x^F from left to right. It splits the stream at EOS, detokenizes, parses the header, and rebuilds the content from the body and the earlier version. Detokenization is exact: when building the masks (§A.3.4), we re-tokenize every record and check it byte for byte. Since $D \mapsto x^F$ is injective, $-\sum_j \log_2 p(x_j \mid x_{<j})$ is a code length for D, so representations of the same D can be compared directly.

Shared and charged information. The following is shared by all arms and not charged: the rendering and decoding code, the tokenizer, the training code and hyperparameters, the random seeds, the warm-start checkpoint (trained only on text written before the stream begins), the selected peak learning rate (at most 3 bits, Appendix A.1) and N. Everything in the stream is charged, including ids, timestamps, titles, and every BOS and EOS. We report ideal code lengths $-\log_2 p$. Arithmetic coding (Witten et al., 1987) comes within 2 bits of them for the whole stream, under the standard assumption that the decoder reproduces training exactly (Blier & Ollivier, 2018).

Table 4: Compact-diff forms. a is the previous version and b the new one. The first character of a body selects its form. In the body syntax, typewriter characters are written literally and italic letters are fields: p is a position in a (counting from 0), n, k and m are numbers of characters, and s is new text. Positions and counts are written in base 62. All examples edit $a =$ `the cat sat` and were produced by the encoder. Bold marks new text, and ␣ marks a space.

			Example	
Edit	Body	Decodes to	New version b	Body
Whole version				
full text	b	b itself. Only if b is a first version or does not start with a mode character, `!><][=+-*@`	`a dog ran`	`a dog ran`
escaped	`!`b	b itself	`== Cats ==`	`!== Cats ==`
unchanged	(empty)	a itself	`the cat sat`	(empty)
One edit at the start or end				
append	`>`s	a followed by s	`the cat sat` **`down`**	`>␣down`
prepend	`<`s	s followed by a	**`so`** `the cat sat`	`<so␣`
cut end	`]`p	The first p characters of a	`the cat`	`]7`
cut start	`[`n	a without its first n characters	`cat sat`	`[4`
One edit anywhere				
overwrite	`=`p`:`s	s written over a at position p	`the` **`b`**`at sat`	`=4:b`
insert	`+`p`:`s	a with s inserted at position p	`the` **`fat`** `cat sat`	`+4:fat␣`
delete	`-`p`:`n	a minus n characters at position p	`the sat`	`-4:4`
replace	`*`p`,`n`:`s	a with the n characters at position p replaced by s	`the` **`dogs`** `sat`	`*4,3:dogs`
Several edits				
multi-splice	`@`k`,`n`,`m`:`$s\ldots$	One hunk per edit. For each, copy the next k characters of a, skip n, and write the m characters of s. Then copy the rest of a	see below	

Multi-splice example. For text as short as `the cat sat`, full text is always shorter, so we use a longer a: `on the mat and the dog lay` → `on` **`a`** `mat and the dog` **`slept`** gives `@3,3,1:ah,3,5:slept`. That is: copy `on␣`, skip `the`, write `a`; copy 17 characters (`h` in base 62), skip `lay`, write `slept`.

A.3.2 COMPACT DIFFS

Let a be the previous version ($\perp$ if there is none) and b the new one. Offsets are 0-based and written in base 62 (digits `0-9a-zA-Z`), and $M = \{\texttt{!}, \texttt{>}, \texttt{<}, \texttt{]}, \texttt{[}, \texttt{=}, \texttt{+}, \texttt{-}, \texttt{*}, \texttt{@}\}$ is the set of mode characters. Let p be the length of the common prefix of a and b, and q the length of the common suffix of what remains. Then, with $s = b[p : |b| - q]$ and $n = |a| - p - q$,

$$b = a[: p] \cdot s \cdot a[|a| - q :]. \tag{3}$$

The encoder $C(a, b)$ emits the shortest applicable form in Table 4, so $|C(a, b)| \leq |b| + 1$. Hunks of the multi-splice come from line-level `difflib` matching refined to characters, and neighboring hunks are merged whenever that gives a shorter body.

Example. A revision of *Kolmogorov complexity* changes "the length of a shortest computer program" to "the length of the shortest computer program". As a unified diff, this is a five-line hunk of 515 characters. As a compact diff, it is `*1Q,1:the`: at offset $\texttt{1Q}_{62} = 114$, delete one character and insert "the". Including the header, this is 47 tokens instead of 170. Two separate edits, "on the mat and the dog lay" → "on a mat and the dog slept", become `@f,3,1:ah,3,5:slept`: copy 15 characters, drop 3, insert "a"; then copy 17, drop 3, insert "slept"; then copy the rest.

Proposition 1 (invertibility). *The decoder of Table 4 returns b from $(a, C(a, b))$ for all a and b, so the compact-diff stream can be decoded left to right.*

Proof. (i) *The form is recoverable.* Whether $a = \perp$ is known from the header: from the Wikipedia event, or from whether the Stack Exchange (site, post id) pair has been seen before. Otherwise, the first character selects the form, and unmarked text is emitted only when $b[0] \notin M$. (ii) *Parsing is*

Table 5: Masking rules on the 1B unified-diff stream: each rule's share of targets and the bits per document the transformer spends on them. In the compact-diff stream, only the header rules apply (10.7% of targets and 0.3% of bits).

Reason	Masked	Targets	Bits/doc
old	Text of context and deletion lines, copied from the previous version	36.11%	339.25
forced	Line markers fixed by the hunk's remaining old and new line counts and its three context lines	1.12%	5.79
grammar	Diff syntax; hunk-header digits fixed by their bounds (e.g. the new start is $a + \Delta$); field labels	1.95%	0.52
memory	Timestamp digits forced by $t_i \geq t_{i-1}$; the rest of a title once a single remembered title matches; the thread id of a known post	0.64%	2.01
framing	Fixed header text once the prefix rules out alternatives (e.g. `iki:` after `W`)	2.34%	0.15
index	The global id, which is the previous id plus one	1.23%	0.24
boundary	`BOS` after `EOS`; `EOS` once no further hunk fits	0.22%	0.08
total		43.60%	348.04

unambiguous. Base-62 digits exclude the delimiters "`,`" and "`:`", the trailing text of a single-edit form ends at the end of the body, and each multi-splice text field carries its own length. (iii) *Each form decodes to b.* Each single-edit form writes one case of equation 3. For example, an append is the case $n = 0$ and $p = |a|$, so $b = a\,s$. Merging hunks (u_j, v_j, s_j) and $(u_{j+1}, v_{j+1}, s_{j+1})$ into $(u_j,\ v_{j+1},\ s_j\, a[v_j : u_{j+1}]\, s_{j+1})$ preserves $b = a[: u_1]\, s_1\, a[v_1 : u_2] \cdots$. Finally, a precedes b in the stream, so it has already been decoded.

A.3.3 Compression accounting

Each window is scored by the weights from before its update, and its first token is an input only. The total bits B_F are the sum of $-\log_2 p$ over all other tokens. We report B_F in three units:

- **Bits per document**, B_F/N. We use this whenever the representations differ (formats, re-diffed orders and masking). At a given size, N is the same for every representation (Table 1).
- **Bits per byte**, B_F divided by the UTF-8 bytes of the rendered records (excluding `BOS` and `EOS`). We use this only within a single rendered stream. Across formats it is misleading: at 1B, full texts reach 0.248 BPB and compact diffs 1.004 BPB, yet they cost 1,564 and 722 bits per document.
- **Masked bits per byte**, the masked bits of §A.3.4 divided by the bytes of the unmasked unified-diff stream.

A.3.4 Masking knowable tokens

A position j is *knowable* if computable functions of the prefix give a decision $\kappa(x_{<j}) \in \{0, 1\}$ and a value $\phi(x_{<j})$ such that $\kappa(x_{<j}) = 1 \Rightarrow x_j = \phi(x_{<j})$. The masked loss is the sum of $-\log_2 p$ over scored positions with $\kappa = 0$. The rules in Table 5 read only the current record so far, the previous record's timestamp, the titles seen so far, and the already-decoded previous version. They label characters, and a token is masked only if every byte it covers is masked. Inserted text, unforced markers and numbers, new titles, and all compact-diff bodies remain scored. Every structural assumption is asserted during labeling, so a violation would raise an error instead of producing a wrong mask. Figure 5a shows the rules applied to one revision.

Proposition 2 (the masked loss is a code length). *Let $q(x_j \mid x_{<j}) = \mathbf{1}[x_j = \phi(x_{<j})]$ where $\kappa(x_{<j}) = 1$, and let q equal the model's p otherwise. Then q is a prequential code for x^F whose length is the masked loss.*

Proof. At every position, q is a distribution over the vocabulary, and it costs 0 bits wherever $\kappa = 1$. The decoder can compute κ and ϕ from the prefix, fill in the masked tokens, and then train on the full window exactly as the encoder did.

Table 6: Bits per document on the 1B stream before and after masking (seed 0). At 10B, masking lowers the transformer from 1,030.3 to 614.3 bits per document, compared with 644.2 for the compact diff.

Arm	All	Masked	Share of bits on knowable tokens
Transformer, unified diff	1,047.6	699.6	33.2%
Retrieval + replay recent	746.6	652.9	12.6%
Retrieval + replay all	732.0	643.2	12.1%
Transformer, compact diff	722.1	719.9	0.3%

The masked loss is therefore an achievable code length, not an oracle bound. On the 1B stream, the unified diff with rule-based copying costs 699.6 bits per document, which is less than the 722.1 of the best compact-diff run (Table 6).

Evaluation-time masking. Masking re-sums the stored per-token losses of the runs in the main text, without retraining. The models are still trained on, and condition on, every token. Training with the masked objective would also give a valid code by Proposition 2, and might do better, but we do not run it.

A.4 Learning rate schedulers

Prequential training has no natural end, which suggests schedules that never read the horizon T, the total number of updates (Defazio et al., 2024; Defazio, 2026). We compare such horizon-free schedules against horizon-dependent ones on the 316M stream, warm-started as in Appendix A.1. We report cumulative prequential bits/byte at two horizons: the first 3,050 updates (100M tokens) and the full stream (316M).

Schedules. Every schedule warms up linearly for 20 updates, and every decaying schedule follows a cosine down to $0.3\times$ the peak. The horizon-dependent schedules are *cosine*, which decays over the whole run, and *WSD*, which decays over the last 90% of the run. The horizon-free schedules are *constant*; *inverse-sqrt*, which decays as $1/\sqrt{t}$; *cyclic cosine*, which restarts its decay every 156 updates; *WSD-S* (Wen et al., 2024), which restarts every 312 updates and decays over the last 20% of each cycle; *WDS*, which decays once over 1,250 updates and then holds its floor; and *schedule-free* AdamW (Defazio et al., 2024). The numbers in the labels of Figure 8 give these lengths in steps of 2,048 tokens, 16 per update.

Tuning and averaging. We screen every schedule at peaks $\{0.5, 1.5, 3, 6\} \times 10^{-4}$ on the first 100M tokens with one seed. All arms share the seed and warm start, so their differences are paired. We then fix the peak at 1.5×10^{-4}, the best peak of an earlier sweep, and test weight averaging. It scores each window with an EMA of the weights, $\bar{\theta}_t = \beta\,\bar{\theta}_{t-1} + (1-\beta)\,\theta_t$, with $\beta = 0.05$ for WSD and WSD-S and $\beta = 0.5$ for constant. Schedule-free scores its averaged iterate by construction. At this peak, WSD + EMA, constant and WSD-S run four seeds at both horizons, except WSD + EMA at 316M, which runs one. Finally, we ablate the WSD decay fraction ($\{10, 25, 50, 90\}\%$) and floor ($\{0.1, 0.3\}$), and grid WDS over decay lengths of 156 to 2,500 updates, floors of $\{0.1, 0.3, 0.5\}$ and peaks of $\{1, 1.5, 2.25\} \times 10^{-4}$.

Results. At each schedule's best peak, cosine and WSD reach 0.767 and 0.768 bits/byte at 100M (Figure 8). The horizon-free schedules do worse: constant and WSD-S reach 0.777, cyclic cosine 0.787, and inverse-sqrt and schedule-free 0.800 and 0.801. Cosine and WSD are best at the largest peak we screened, so their numbers are, if anything, pessimistic. The exception is WDS, which ties cosine at 0.767. WDS never reads T, but it decays once, early, over a length that is tuned at this horizon. At the fixed peak, the best WDS configuration in the grid trails constant by 0.004 bits/byte at 100M and by 0.012 at 316M, but we did not rerun WDS at 316M at its best screened peak.

Averaging does not close the gap. EMA moves constant and WSD-S by at most 0.002 bits/byte. At the fixed peak, which is below the best peak of every decaying schedule, the schedules are close. WSD + EMA trails constant by 0.001 ± 0.002 bits/byte at 100M (paired over four seeds) and leads by 0.0006 at 316M (one seed), and WSD-S trails constant by 0.003–0.005 on every seed at both horizons. The shape of WSD matters little: at the fixed peak, all decay fractions and floors land

between 0.790 and 0.796 bits/byte. We therefore use WSD throughout, with the stable phase and floor given in Appendix A.1.

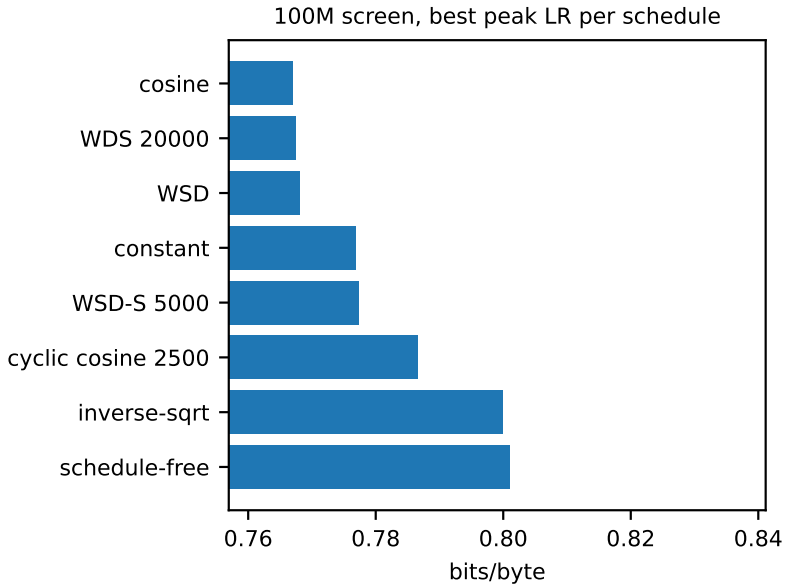


Figure 8: **Horizon-dependent schedules are best.** Cumulative bits/byte over the first 100M tokens of the 316M stream, for each schedule at its best peak in $\{0.5, 1.5, 3, 6\} \times 10^{-4}$ (one seed). Cosine and WSD read the horizon. WDS is the only horizon-free schedule that matches them.

A.5 Learning rate stability of warm start and fresh initialization

We sweep the WSD peak learning rate from 2.4×10^{-4} to 7.68×10^{-3} in factors of two on the 316M stream, with five seeds per learning rate, both warm-started from the warm-start checkpoint and from a random initialization (Figure 9). Up to 3.84×10^{-3} the warm start is both better and much more stable: its best mean is 0.708 BPB at 4.8×10^{-4}, and up to 9.6×10^{-4} its seeds have an SD of about 0.008 BPB. Only at the highest peak, 7.68×10^{-3}, does fresh initialization do better. Fresh initialization is unstable around its optimum. Its best mean is 1.188 BPB at 9.6×10^{-4}, and its best single run (1.066 BPB) is at 1.92×10^{-3}, but at that learning rate the five seeds span 1.07 to 1.53 BPB (SD 0.20), the largest spread in the sweep.

A.6 Frozen and online cumulative losses

Figure 2b plots the difference between two cumulative losses. Figure 10 shows both curves. The frozen model is the pre-pretrained checkpoint, which scores the stream without any updates. The online model starts from the same checkpoint and trains on each window after scoring it (WSD, peak 4.8×10^{-4}). The frozen loss stays near 1.9 bits/byte and rises slightly late in the larger streams, which move further from the checkpoint's data. The online loss falls steadily. At the end of the stream, online training reduces bits/byte by 62.1% at 100M (1.926 to 0.730), 63.7% at 316M (1.942 to 0.704) and 64.8% at 1B (1.979 to 0.697).

A.7 Shuffled order and staleness

Shuffle variants. Each variant reorders the stream with one fixed random permutation (seed 0). *Shuffled windows* cuts the stream into its 32,768-token training windows and trains on them in random order. Documents and window boundaries are unchanged, so the model scores exactly the same tokens. *Shuffled documents* permutes the documents before the stream is cut into windows. Each document keeps its rendering, so a Wikipedia or Stack Exchange diff still refers to its chronological predecessor, which may now come later in the stream. The tokens are again unchanged. *Shuffled both* applies both permutations. *Shuffled documents + rediff* keeps the shuffled document order but

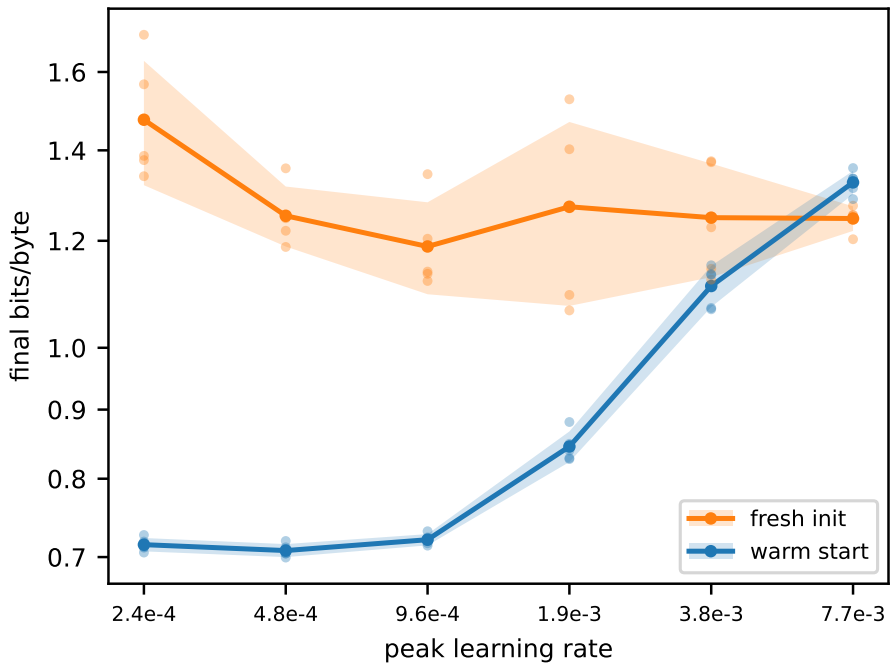
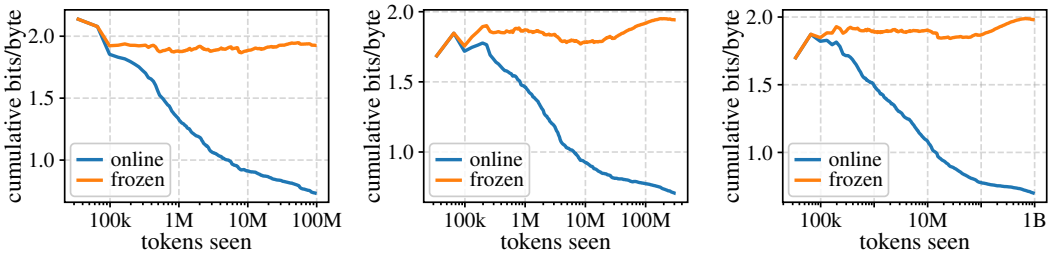


Figure 9: Final BPB on the 316M stream against the WSD peak learning rate, warm-started from the warm-start checkpoint and from a random initialization. Lines are the mean over five seeds, bands are ± 1 SD, and each dot is one seed (overlapping seeds appear darker).



(a) 100M (b) 316M (c) 1B

Figure 10: Cumulative bits/byte against tokens seen for the frozen pre-pretrained checkpoint and the model trained online from it, on each stream. Mean of two seeds.

re-renders each Wikipedia page and Stack Exchange post in full the first time it appears, and as a diff against the most recently seen version after that. Every diff then refers to text the model has already seen, but the diffs are much larger: the 1B stream grows to 6.3B tokens. We therefore compare all variants in bits/article, the total code length divided by the number of documents, which is the same for every variant.

Every variant runs WSD at a peak of 4.8×10^{-4} with two seeds on the 100M, 316M and 1B streams. In Figure 11, the sequential and shuffled-windows lines instead use the baselines of Appendix A.1, each at its selected peak and up to 10B. On the 1B stream, shuffling the windows costs 2.9% bits/article (1,046 to 1,077), shuffling the documents 18.3% (1,238), and rediffing them 147% (2,587). Once the documents are shuffled, also shuffling the windows changes bits/article by at most 0.1% at every size (Figure 11).

Measuring staleness. A staleness run trains exactly like a baseline, one window per update. In addition, before update t trains on window w_t with weights θ_{t-1}, it scores the future windows w_{t+k} with the same θ_{t-1} for $k \in \{1, 2, 4, \ldots, 64\}$, without training on them. Window w_{t+k} is thus predicted by a model that has missed the k most recent windows ($k \times 32{,}768$ tokens, or 2.1M at

$k = 64$), and $k = 0$ is the ordinary prequential loss. All staleness levels come from the same training trajectory. We average over windows 64 onwards in training order, which every level scores. Each window's loss is divided by that window's own bytes and the windows are weighted equally, so the $k = 0$ average differs slightly from the stream-level bits/byte of the baselines (0.694 against 0.697 at 1B). We run the sequential stream, shuffled windows and shuffled documents on the 1B stream at a WSD peak of 4.8×10^{-4}, with two seeds each.

Relation to batching. With batch size B, all B windows of a batch are scored with the weights from before the batch, so they are stale by $0, 1, \ldots, B-1$ windows, or $(B-1)/2$ on average. Staleness k is therefore the delay of the last window in a batch of $k+1$. It is not the loss of a batch-B run, which also takes B times fewer and larger steps and needs its own learning rate (McCandlish et al., 2018). Our measurement holds the trajectory fixed and isolates the cost of the delay. It does not capture how larger batches change the trajectory itself.

Results. Staleness is costly only for the sequential stream (Figure 12b). Its bits/byte rises from 0.694 at $k = 0$ to 0.701 at $k = 8$ (+1.1%) and 0.714 at $k = 64$ (+2.9%). Over the same range, shuffled windows rises by 0.2% (0.715 to 0.716) and shuffled documents by 0.3% (0.821 to 0.823). At $k = 64$ the sequential stream keeps only 0.002 of its 0.021 bits/byte advantage over shuffled windows. The seed standard deviation is below 0.003 bits/byte in every cell.

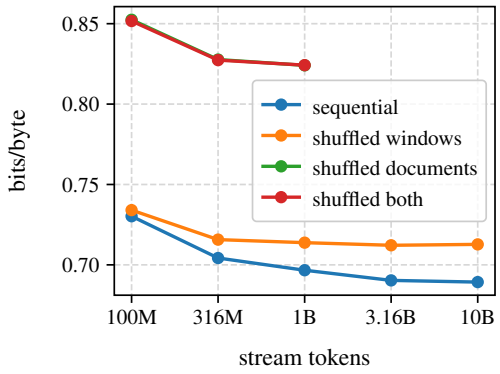
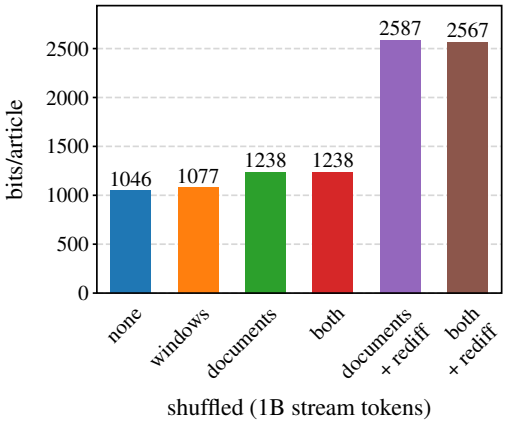
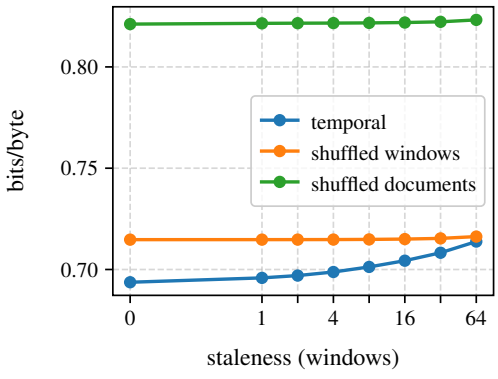


Figure 11: Final BPB against stream size for the sequential stream, shuffled windows, shuffled documents, and both shuffled.



(a) Shuffled order in bits/article, including shuffled + rediff



(b) Staleness, including shuffled documents

Figure 12: **Shuffling documents costs far more than shuffling windows, and only the sequential stream suffers from staleness.** Left: bits/article on the 1B stream for every shuffle variant, including shuffled + rediff. Right: mean bits/byte on the 1B stream when each window is scored with weights k windows stale. Seed means.

A.8 REPLAY AND EXACT-MATCH RETRIEVAL

Neither method changes which tokens are scored. The stream is split into consecutive, non-overlapping windows $w_1, w_2, \ldots$ of $L = 32{,}768$ tokens. At update t we first score w_t with the current parameters θ_{t-1} and then take one optimizer step, and the prequential loss is the sum of these scores. Any prediction that depends only on $w_{<t}$ and on earlier tokens of w_t is therefore a valid code length. Table 7 lists all hyperparameters.

Replay. At update t we draw k distinct windows uniformly at random from a pool R_t of earlier windows. For `replay all` the pool is all of $w_1, \ldots, w_{t-1}$. For `replay recent` it is the last 512 windows $w_{t-512}, \ldots, w_{t-1}$ (16.8M tokens). The update minimizes

$$\tfrac{1}{2}\,\bar{\ell}(w_t) + \tfrac{1}{2}\,\bar{\ell}(w_{r_1} \cup \cdots \cup w_{r_k}),$$

where $\bar{\ell}$ is the mean next-token cross-entropy over the tokens of its argument. The current window therefore gets half the weight for every k. Only w_t is scored. Replayed windows only change the parameters used for later windows, so replay adds compute but no code length. Replay starts once $|R_t| \geq k$, and replayed windows are trained with the language model alone, without retrieval. Each update processes $1 + k$ windows, so we run replay up to the 3.16B stream and not at 10B.

Exact-match retrieval. We keep a cache $c_1, \ldots, c_N$ of every token scored so far, in stream order. A hash table maps each distinct 16-token string to the (up to) four most recent positions j that it precedes. That is, $c_{j-16:j-1}$ is the string and c_j is the token that followed it. For each position i in w_t with at least 16 preceding tokens in the window, we look up the context $x_{i-16:i-1}$. If it has been seen, i is *matched*, and each candidate j gets a match length m_j: the longest common suffix of $x_{<i}$ and $c_{<j}$, capped at 512 tokens and at the start of w_t. The retrieval distribution is uniform over the continuations of the longest-matching candidates:

$$p_{\mathrm{ret}}(v \mid x_{<i}) = \frac{|\{\, j \in J^\star : c_j = v \,\}|}{|J^\star|}, \qquad J^\star = \arg\max_j m_j.$$

We query the cache with w_t before inserting it, so retrieval only sees $w_{<t}$. Repetitions within the current window are left to attention.

Gate. The model's prediction is a mixture of the language model and retrieval with a learned per-token weight:

$$P(x_i \mid x_{<i}) = (1-\lambda_i)\, P_\theta(x_i \mid x_{<i}) + \lambda_i\, p_{\mathrm{ret}}(x_i \mid x_{<i}), \qquad \lambda_i = 0.95\,\sigma\big(u^\top \mathrm{SiLU}(W h_i + b) + c\big),$$

with $\lambda_i = 0$ at unmatched positions. Here h_i is the backbone's final normalized hidden state (the input to the LM head) and $W \in \mathbb{R}^{64 \times d}$. We initialize $u = 0$ and set c so that $\lambda_i = 0.05$. Capping λ_i at 0.95 bounds the extra cost of a wrong retrieval by $\log_2 20 \approx 4.3$ bits. The mixture is a proper distribution that uses only causal information, so its negative log-likelihood is the code length we report. It is also the training loss for the backbone and the gate. Since the gradient of the mixture loss with respect to $\log P_\theta$ is scaled by $(1 - \lambda_i)P_\theta/P$, the backbone gets less training signal on tokens that retrieval already predicts well. The gate has its own optimizer (Table 7).

Implementation. Retrieval runs on the CPU and overlaps with the GPU step through data prefetching. We index 16-token strings by rolling hashes and verify every hit against the stored tokens, so hash collisions never merge different strings. Positions are stored in 40 bits, and the cache for the 10B stream uses about 187 GiB of host memory.

A.9 REVISION FORMATS

Any injective rendering of the records gives a valid code length (Appendix A.3.1), so we can compare how revisions are written. The three formats below change only the bodies of Wikipedia and Stack Exchange revisions. Every format writes first versions in full.

Formats. The *unified diff* is our default (Appendix A.2.2). Each hunk repeats three lines of context around a change and writes every changed line out in full, once as deleted and once as added. The *compact diff* writes the shortest edit from a small set of invertible operations (Appendix A.3.2). The *full article* writes the whole new version at every revision and never refers to the previous one. On the 1B stream, the three formats take 1.00B, 0.50B and 4.38B tokens.

Table 7: Replay and retrieval hyperparameters. The backbone uses the same AdamW and WSD schedule as the baseline in every arm.

Hyperparameter	Value
Replay	
Replayed windows per update k	3 (1, 3, 7 in Figure 4a)
Pool, `replay recent`	last 512 windows
Pool, `replay all`	all earlier windows
Loss weight, current : replayed	1/2 : 1/2
Exact-match retrieval	
Lookup (minimum match) length	16 tokens
Maximum match length	512 tokens
Candidates per 16-token string	4 most recent
Gate	
Architecture	$d \to 64 \to 1$ MLP, SiLU
λ cap / initial λ	0.95 / 0.05
Optimizer	Adam, constant LR 10^{-3}
Gradient clipping / weight decay	1.0 / 0

Setup. All runs use seed 0, temporal order and the recipe of Appendix A.1. The compact diff and full article sweep the peak learning rate over $\{0.25, 0.5, 1, 2, 4\} \times 4.8 \times 10^{-4}$, and we report the best peak. The unified diff reuses the seed-0 baseline run at 4.8×10^{-4}. At 3.16B and 10B, the selected peak of 2.4×10^{-4} has at most 1.0% lower seed-mean bits/byte, far below the gaps between formats. We run the full article only up to 1B, where it already takes 4.38B tokens. Formats produce streams of different lengths, so we compare them in bits per document (Appendix A.3.3).

Results. The compact diff beats the unified diff at every size, and the gap widens with scale (Figure 13). It saves 23.7% of bits per document at 100M (855.5 against 1,121.0), 31.1% at 1B (722.1 against 1,047.6) and 38.3% at 10B (644.2 against 1,043.8). From 1B to 10B, the unified diff barely improves (1,047.6 to 1,043.8), while the compact diff keeps improving (722.1 to 644.2). The full article is the worst format, and it gets worse with scale, from 1,405.6 bits per document at 100M to 1,563.8 at 1B, which is 2.2 times the compact diff. Even with the previous version written out verbatim, the model pays heavily to repeat it, so the loss grows with the amount of redundant text it must predict. Masking the knowable tokens of the unified diff, which removes this redundancy without changing the stream, gives the lowest losses of all (Appendix A.3.4).

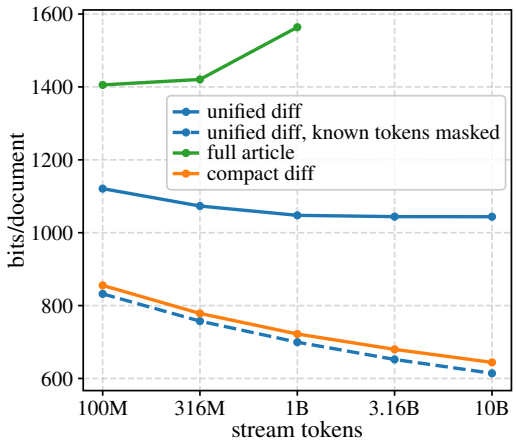


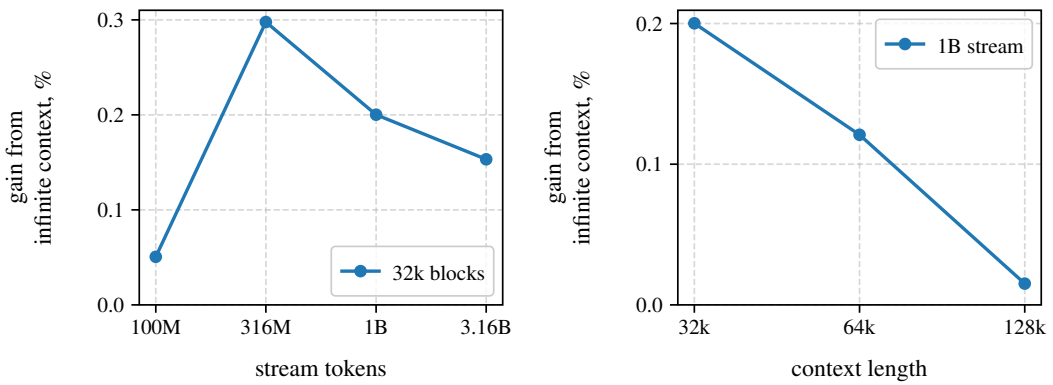
Figure 13: Bits per document against stream size for unified diffs (with and without the known tokens masked), compact diffs, and the full article at every revision, at the best peak learning rate per format and size (unified diff: 4.8×10^{-4} at every size, seed 0).

A.10 INFINITE CONTEXT FOR GATED DELTANET

A Gated DeltaNet layer compresses its entire past into a fixed-size state, so unlike attention it can in principle read the whole stream at constant cost per token. We test this by carrying the state across windows.

Setup. At the end of each window, every Gated DeltaNet layer hands its recurrent state to the next window: the fp32 128×128 matrix state of each of its 8 heads, and the last three inputs to its causal convolution. The state is detached, so gradients stay within the current window, as in truncated backpropagation through time (Williams & Peng, 1990). It starts at zero at the beginning of the stream and is computed with the weights from before each update. The full-attention layers stay window-local, so earlier windows reach the model only through the carried state and the weights. Each infinite-context run is paired with a finite-context run that resets the state at every window. The pair shares the warm start (itself trained with window-local state), seed, window length and peak learning rate (4.8×10^{-4}, the hybrid's best peak on the 316M stream), so the two runs differ only in the carried state. We run two seeds at 32k on the 100M to 3.16B streams, and one seed at 64k and 128k on the 1B stream. The infinite-context runs also score the first token of each window, which the finite recipe skips. This adds one token in 32,768 and can only increase their bits/byte. Carrying the state costs little: at 32k the infinite-context runs process 28–29k tokens/s, within the range of the finite runs.

Results. The carried state lowers bits/byte in all ten pairs, but only slightly (Figure 14). At 32k the gain is 0.05% on the 100M stream, 0.30% at 316M, 0.20% at 1B and 0.15% at 3.16B, with no trend in stream size. The gain also shrinks as the window grows, to 0.12% at 64k and 0.02% at 128k on the 1B stream. For comparison, doubling the window from 32k to 64k alone lowers bits/byte by 1.1% (0.6485 to 0.6416). This is consistent with the gated state retaining mostly recent context, which a longer window already covers.



(a) By stream size, 32k blocks

(b) By block length, 1B stream

Figure 14: **Carrying the Gated DeltaNet state across windows helps little.** Gain from infinite context for the hybrid Gated DeltaNet, as the percentage reduction in final bits/byte of the run that carries its state across blocks over the matched run that resets it. Seed means. Left: against stream size at 32k. Right: against block length on the 1B stream, where 64k and 128k ran one seed.