TiC-LM: A Web-Scale Benchmark for Time-Continual LLM Pretraining

Jeffrey Li[1*†] **Mohammadreza Armandpour**[2†] **Iman Mirzadeh**[2]

Sachin Mehta[°] **Vaishaal Shankar**[°] **Raviteja Vemulapalli**[2] **Samy Bengio**[2]

Oncel Tuzel[2] **Mehrdad Farajtabar**[2] **Hadi Pouransari**[2] **Fartash Faghri**[2‡]

[1]University of Washington, [2]Apple

jwl2162@cs.washington.edu,fartash@apple.com

[*]Work done during an internship at Apple. [°]Work done while at Apple. [†]Equal contribution [‡]Project lead

Abstract

Large Language Models (LLMs) trained on historical web data inevitably become outdated. We investigate evaluation strategies and update methods for LLMs as new data becomes available. We introduce a web-scale dataset for *time-continual pretraining* of LLMs derived from 114 dumps of Common Crawl (CC) – orders of magnitude larger than previous continual language modeling benchmarks. We also design time-stratified evaluations across both general CC data and specific domains (Wikipedia, StackExchange, and code documentation) to assess how well various continual learning methods adapt to new data while retaining past knowledge.[1] Our findings demonstrate that, on general CC data, autoregressive meta-schedules combined with a fixed-ratio replay of older data can achieve comparable held-out loss to re-training from scratch, while requiring significantly less computation (2.6×). However, the optimal balance between incorporating new data and replaying old data differs as replay is crucial to avoid forgetting on generic web data but less so on specific domains.

1 Introduction

Large language models (LLMs) rely on massive amounts of data, most of which comes from large-scale web-crawls run over the past 10–20 years. Common Crawl (CC), the most well-known source of such data, has been active since 2007 and continues to release monthly dumps of data. While typically LLMs are trained from scratch on many (or all) previous dumps jointly (Penedo et al., 2023; Li et al., 2024a), they also suffer from the knowledge cutoffs of their training sets causing their performance to deteriorate on newer data (Cheng et al., 2024). Combined with the vast costs of re-training LLMs from scratch, a natural question is how LLMs can best be *continually* reused and updated over many months and years.

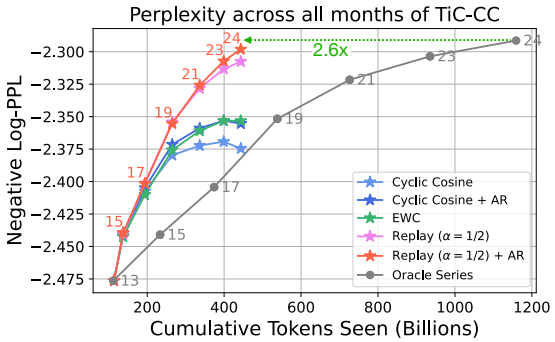


Figure 1: **Replay allows for matching repeated from-scratch training.** Each line traces a continually trained 3B model where the annotations indicate the data cutoff year of the checkpoint (e.g., "24" = 2024). We find that combining autoregressive (AR) learning rate schedules and data replay (red) can nearly match the perplexity on all months achieved by the Oracle series which re-trains from scratch every two years (gray), despite requiring 2.6× less compute. Meanwhile, methods that only modify the optimizer (blue) or loss (green) insufficiently prevent forgetting and plateau.

To study this question, our take is that a benchmark of appropriate scale and scope is a key prerequisite missing from the current literature. In particular, many prior works on continual language modeling train and evaluate on single domains such as Wikipedia, News, or Twitter/X (Jang et al., 2022a; Liška et al., 2022; Luu et al., 2022). However in practice, LLMs are trained on *general* web-scale data (in which many implicit domains and their relative presence both evolve over time) with the explicit goal of performing well across many types of tasks. Further, while more recent efforts (Gupta et al., 2023; Parmar et al., 2024; Ibrahim et al., 2024) do study continual LLM training at web-scale, their setups do not focus on *temporal* distribution shifts and contain generally less than three training rounds. This limits their potential generalizabiltiy for studying time-continual learning across longer horizons in a truly lifelong sense.

Taking inspiration from TiC-CLIP (Garg et al., 2024), our work aims to address these gaps by

[1]Our code: https://github.com/apple/ml-tic-lm

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 32231–32273

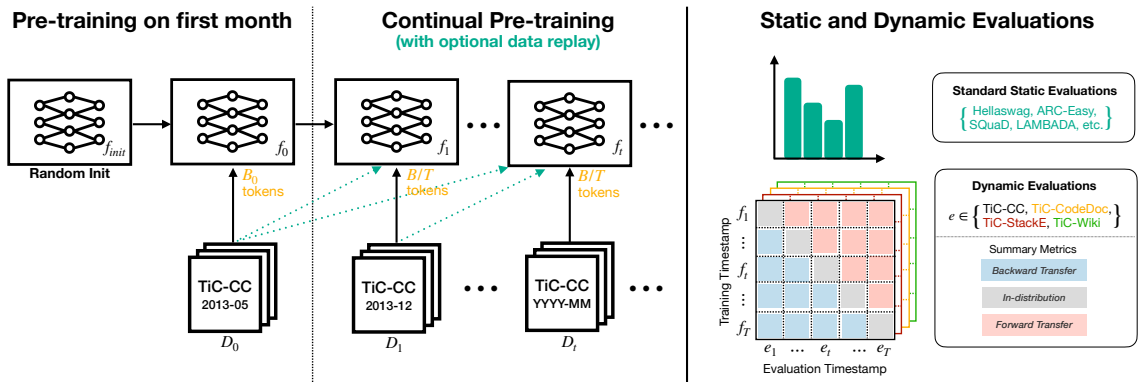


Figure 2: **The TiC-LM benchmark.** We simulate a setup where each Common Crawl dump $D_0, \cdots, D_T$ is revealed one-at-a-time. An LLM f_0 is first pre-trained for B_0 tokens on the initial month D_0 and then continually updated for a fixed budget of B/T tokens in each following month (optionally replaying older data). The goal is for each monthly model $f_1, \cdots, f_T$ to perform well on both standard static downstream tasks as well as dynamic evaluations that evolve over time, requiring the balance of learning (gray/red) with preventing forgetting (blue).

introducing the **TiC-LM** (**Ti**me-**C**ontinual Learning of **L**anguage **M**odels) **benchmark** (see Fig. 2). Our setup centers on TiC-CommonCrawl (TIC-CC), a massive time-stratified set of training and held-out evaluation data created using 114 months (May-2013 to July-2024) of CC data, where during training, each month is revealed only one at a time. In total, TIC-CC contains 2.9T possible training tokens spread among the 114 timesteps, providing 100× more potential tokens and 10× more timesteps compared to prior time-continual LLM benchmarks. For evaluation, TiC-LM also provides several domain-specific *dynamic* evaluations sourced from diverse domains including TiC-Wikipedia (TIC-WIKI), TiC-StackExchange (TIC-STACKE), and TIC-CODEDOCS.

Using our benchmark, we run over 150 experiments to evaluate the effectiveness of different optimization, replay, and regularization-based continual learning strategies. These results provide insights into the following key questions:

- *Can continual pretraining match periodic re-training at lower cost?* We find that a mix of learning rate and data replay strategies allows us to be competitive with a series of models that retrains from scratch every two years requiring 2.6× less total compute (see Fig. 1). However, trade-offs remain between continual methods and re-training across domains and evaluations.
- *Is forgetting a challenge when continually pre-training on web-data?* We observe that on general web-data in TIC-CC, older CC dumps are significantly forgotten when only training on new data and replay is essential to retain performance on these earlier dumps (see Fig. 3).
- *Is the impact of forgetting domain-dependent?* Forgetting older CC dumps need not always be detrimental. Replaying old data can actually hurt when evaluating on rapidly evolving domains like StackOverflow and PyTorch, while still benefiting more stable ones where older dumps are more useful such as Math and NumPy (see Fig. 4).

We release all code for our training and evaluation setups. Our hope is that the broader research community will use these assets to further realize the potential of continual LLM pretraining.

2 Related Work

Learning new capabilities from multiple, sequentially observed, distributions has long been an active area of ML research (Wang et al., 2024). More recently, several works have studied this setting for LLMs (Wu et al., 2024), targeting improvements on: (1) general capabilities (via higher-quality datasets) (Parmar et al., 2024; Ibrahim et al., 2024; Gupta et al., 2023); (2) specific domains (Jin et al., 2022; Gururangan et al., 2020; Chen et al., 2024); (3) newer data as the world evolves (Jin et al., 2022; Jang et al., 2022b,a; Lazaridou et al., 2021; Nylund et al., 2024; Loureiro et al., 2022; Qin et al., 2022; Liška et al., 2022). Works in this third category have shown that in many domains, the performance of LLMs decays as training and evaluation sets grow farther apart in time, motivating the need for methods to *efficiently* adapt to new data while retaining existing knowledge. Our work scales up these previous efforts to more closely match current LLM training practices. While older works typically focus on continual training and evaluation

Table 1: Comparison of TiC-LM (bottom) with previous continual learning studies for LLMs. The "Temporal" column refers to whether the continual training rounds are defined across temporal distribution shifts. Training set size is in terms of tokens unless otherwise specified (i.e., Art. = Articles, QA = Question-Answer pairs).

Benchmark	Domain	Temporal	Steps	# Train
Gupta et al. (2023)	Web	✗	2-3	297B
Parmar et al. (2024)	Web	✗	2	3.8B-500B
Ibrahim et al. (2024)	Web	✗	2–7	100B
Chrono. Tweet (2022)	Science, Tweets	✓	4	25M
TempEL (2022)	Wiki	✓	10	<138k Art.
TemporalWiki (2022a)	Wiki	✓	4	23B
StreamingQA (2022)	News	✓	12	99k Art.
EvolvingQA (2024)	Wiki	✓	6	390k Art.
TIQ (2024)	Wiki	✓	—	6k QA
TAQA (2024)	Wiki	✓	—	9k QA
Luu et al. (2022)	Science, News Reviews, Tweets	✓	4–7	695k Art.
TIC-CC	Web	✓	114	220B-440B (up to 2.9T)
TIC-WIKI	Wiki	✓	62	—
TIC-STACKE	Diverse QA	✓	8–170	—
TIC-CODEDOCS	Code	✓	11–16	—

over individual sources (e.g., news, Wikipedia, and social media) and $\leq$10 timesteps, we consider training on a *generic web-crawl* (i.e., Common Crawl) spanning 114 different months. In turn, the generality of our training data allows us to go beyond single-domain evaluations. Table 1 summarizes our proposed datasets compared with the most related continual benchmarks. With 2.9T tokens, TIC-CC is the *largest and most diverse* continual learning benchmark for language modeling. We provide an extended discussion of related works in Appx. E.

3 TIC-CC: over 10 years of web data

We create a large English *time-stratified* dataset of 2.9T tokens based upon Common Crawl (CC), a free and open corpus of web-crawled data that has been online since 2007. We collect all dumps between May-2013 and July-2024, resulting in 114 corresponding splits that we refer to by the month of their release date. For each split, we then apply a pre-processing pipeline based on that of DataComp-LM (Li et al., 2024a) to create a corpus representative of existing pre-training datasets. Notably, to retain causality, we do not perform any operations on older months that depend on future months.

Data processing. We build upon the assets from DataComp-LM (Li et al., 2024a), starting with DCLM-Pool (Li et al., 2024a), which contains all CC dumps up to Dec-2022 and parsed to extract plaintext from webpages via the `resiliparse` library (Bevendorff et al., 2018, 2021). We split this data by month and reuse the same download and processing scripts to extend DCLM-Pool until July-2024. Next, we follow DCLM-Baseline's pipeline by applying heuristic filters from RefinedWeb (Penedo et al., 2023) and a fuzzy-deduplication step which we modify to run only *within* each month rather than across all months. Alternatively, as in TiC-CLIP, one could deduplicate newer months against older ones. Here, we do not do this by default given that it may not always be helpful (e.g., it may remove updated or edited pages where a few key facts have changed but most text remains the same). We perform an initial exploration of cross-month deduplication in Appendix D.7 and overall allow for exploring the benefits/pitfalls of such data-centric interventions as part of method design. Finally, we do not use the final classifier-based filter in DCLM-Baseline, as this classifier was trained on data from all months, violating causality. For more details about the data pipeline, see Appx. A.

In Fig. 6 (left), we show the number of tokens we have for each month of TIC-CC. In total, the dataset spans 29T tokens, with individual months ranging between 100B to 500B tokens. Our experiments train on subsets of 220-440B tokens from a single global shard that contains 2.9T while future work can expand to the full 29T.

4 Evaluations

In this section, we discuss various time-continual evaluations that are designed both with and independent of TIC-CC. As our focus is on pretraining, we focus on evaluations without instruction-tuning.

Perplexity metrics. We employ three distinct perplexity metrics for different evaluations:

$$\text{ppl}_{\text{token}} = \exp\left(\frac{\sum_{d\in\mathcal{D}}\sum_{t\in T_d} -\log P(t|c_{<t})}{\sum_{d\in\mathcal{D}}|T_d|}\right) \quad (1)$$

where $\mathcal{D}$ is a set of pages, T_d is the set of tokens in page d, and $c_{<t}$ is the context prior to token t.

$$\text{ppl}_{\text{answer}} = \frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}} \exp\left(-\log P(a_q|c_q)\right) \quad (2)$$

where $\mathcal{Q}$ is a set of question-answer pairs, a_q is the gold answer for question q, and c_q is the context.

$$\text{ppl}_{\text{noun}} = \exp\left(\frac{\sum_{d\in\mathcal{D}}\sum_{n\in N_d} -\log P(n|c_{<n})}{\sum_{d\in\mathcal{D}}|N_d|}\right) \quad (3)$$

where N_d is the set of proper noun tokens found by a PoS tagger (Honnibal and Montani, 2017) in page d, and $c_{<n}$ is the context prior to noun n.

4.1 TiC-CommonCrawl (TIC-CC)

We compute $\text{ppl}_{\text{token}}$ on three monthly subsets of our CC data which were held out from training:

- TIC-CC: Held-out pages coming from the full distribution for each month of TIC-CC.
- TIC-CC-WIKI: Pages in TIC-CC from English Wikipedia and Simple English Wikipedia
- TIC-CC-NEWS: Pages in TIC-CC from a set of news sites based on WMT competitions (Barrault et al., 2020).

These evaluations align exactly with the training objective and data, providing direct signal on how well different months and subsets were learned.

4.2 TiC-Wikipedia (TIC-WIKI)

TIC-CC-WIKI in Sec. 4.1 uses a representative subset of Wikipedia pages found in CC dumps. In contrast, TIC-WIKI aims to (1) comprehensively build examples using complete Wikipedia dumps; (2) isolate changed/unchanged factual knowledge through focusing on proper nouns (ppl_{noun}) as opposed to all tokens ($\text{ppl}_{\text{token}}$), following (Jang et al., 2022a; Lazaridou et al., 2021). To construct TIC-WIKI, we build upon TemporalWiki (Jang et al., 2022a), expanding its coverage from four months to a decade (2014–2024) while improving the parsing of Wikipedia (see Appx. B.1). This results in TIC-WIKI-Diff and TIC-WIKI-Unchanged which capture the changed and unchanged facts across dumps, respectively.

4.3 TiC-StackExchange (TIC-STACKE)

We design another evaluation based on the historical data from StackExchange. StackExchange has 182 communities that share knowledge by posting questions and answers. We measure answer-perplexity ($\text{ppl}_{\text{answer}}$) on high-quality answers from selected sites by collecting answers that have been accepted by the question author (using the accepted answer timestamp to bin examples by month). The resulting evaluation contains examples from 2008–2024 (see Appx. B.2 for more details).

4.4 TIC-CODEDOCS

Our TIC-CODEDOCS evaluation is based on the official documentation from major releases of NumPy (Harris et al., 2020) (16 releases from 2017-2024, `v1.13.0` to `v2.1.0`) and PyTorch (Ansel et al., 2024) (11 releases from 2021-2024, `v1.8.0` to `v2.4.0`). We build documentation from each library's Git repository by reverting to release commits and generating HTML documentation from source, which we convert to plain text using `readability.js` to retain only the main content. We evaluate model performance using perplexity ($\text{ppl}_{\text{token}}$) across version snapshots.

4.5 Static downstream evaluations.

We evaluate models on a variety of downstream zero-shot and few-shot tasks suitable for base models. Specifically, we use the CORE average from the DCLM benchmark (Li et al., 2024a) which includes 22 zero-shot and few-shot in-context learning tasks. These evaluations, which include benchmarks such as ARC-Easy (Clark et al., 2018) and Hellaswag (Zellers et al., 2019), assess general capabilities of base models via a variety of world knowledge and natural language understanding tasks. While they are not designed to be time-dependent, we use them to assess whether continually trained models match the general capabilities of models trained on all dumps.

5 Continual Learning Baseline Methods

We now go over the continual methods that we test on our benchmark as well as relevant non-continual baselines. For continual methods, we consider the following three categories: optimization-based, data replay, and regularization.

Optimization-based methods. In non-continual settings, LLMs are often trained with a cosine-decayed learning rate schedule which requires knowledge of total training steps ahead of time. In a continual setup, however, the number of total tokens grows over time and we care about the performance after each month. We benchmark the following optimization approaches in our work:

- *Cyclic Cosine* is the simplest alternative which applies cosine decay *within* each training month using the same maximum learning rate and warmup (Fig. 15, green). This was found to be most effective in TiC-CLIP (Garg et al., 2024).
- *Cyclic Cosine + AR (autoregressive)* uses Cyclic Cosine but also decays the maximum learning rate across rounds, regressed from a global cosine schedule (Fig. 15, blue). It was shown to offer improvements by Roth et al. (2024).
- *Rsqrt* (*reciprocal-*$\sqrt{}$) are *infinite* schedules that decay the learning rate slowly in a global training run and branch off of this trajectory with linear cooldowns (Zhai et al., 2022). To keep training

steps fixed compared to other methods, we follow Roth et al. (2024) and implement a version that maintains only a single trajectory by re-warming up from the previous cooldown.

- *Schedule-Free* is an optimizer proposed by Defazio et al. (2024) which aims to circumvent the need for defining a learning rate schedule by using iterate averaging and has achieved promising results in i.i.d. non-continual settings.

Data replay methods. A classical strategy to prevent forgetting is to mix the current timestep's data with data from earlier timesteps (Lopez-Paz and Ranzato, 2017; Rebuffi et al., 2017; Chaudhry et al., 2018). Defining a replay method therefore boils down to defining the mixture ratios across rounds. Based on TiC-CLIP (Garg et al., 2024), we mostly consider variants of the form:

- For the current timestep t, we allocate a ratio $0 \leq \alpha_t \leq 1$ of the monthly token budget B_t to data from the current month, seeing $\alpha_t B_t$ tokens from that month.
- For previous months, we redistribute the remaining $(1 - \alpha_t)B_t$ tokens equally, i.e., each month contributing $\frac{1-\alpha_t}{t-1} B_t$ tokens to this round's data.

In our setup, B_t is fixed across months. We first try $\alpha_t = 1/t$, which sees an equal number of tokens from all observed months. We also consider the constant value $\alpha_t = 1/2$ which always allocates half the token budget to the current month. In Appendix D, we try other fixed settings of α_t as well as exponentially down-weighting older months based on distance from the current timestep. One potential downside of replay-based methods is the cost of retaining old data, especially if old data expires and needs to be removed. Methods with larger values of α_t are less affected by such limitations. We do not consider such costs in our work (as we assume they are likely to be dominated by training costs).

Regularization-based methods. These methods alter the training objective by adding a regularization term which encourages newer model updates to stay close to the model weights learned after the previous month. Following TiC-CLIP, we try two notable methods: LwF (Li and Hoiem, 2018) and EWC (Kirkpatrick et al., 2017).

- *LwF* adds a KL divergence loss term to penalize differences in model outputs between the previous checkpoint and the current model.
- *EWC* attempts to slow down updates to particular model parameters which are highly influential for performing well on older months as measured by the (approximate) Fisher information matrix.

Because both LwF and EWC involve extra loss terms and model copies, it is important to note that they induce larger GPU memory footprints and run-times compared to optimizer and replay-based methods. That being said, we do not try to adjust the token counts to account for this given that our re-implementations may not be optimally efficient.

Non-continual Oracles. The alternative to continual learning is to simply retrain a model from scratch on an equal number of tokens from all currently available months. We refer to such a model as *Oracle-t* where t is particular cutoff date. We then consider as a baseline a *series of Oracle models* re-trained roughly every two years (i.e., {2013-05, 2015-01, 2017-01, 2019-01, 2021-01, 2023-01, 2024-07}), where for any timestamped evaluation, the most recently trained *Oracle* is always used.

6 Experiments

Training details. We train 1B and 3B parameter models using OpenLM (Gururangan et al., 2023). Unless otherwise indicated, most of our results are for 3B models, where each method sees the same total number of 220B or 440B tokens, equivalent to $4\times$ and $8\times$ the Chinchilla optimal amount. [2] For 1B models, we only train at the 220B token scale to study how model size might affect the results (see Table 2 and Appendix D.8). For continual runs, we further assume that current practitioners are (a) likely to have access to more than enough data to train initial models; (b) unlikely to wait to obtain non-trivial performance. Hence, we front-load the total token budget such that 110B is allocated to an *initial pretraining* on the first month (May-2013). Then, the remaining tokens are split equally among the other 113 continual timesteps. For our *Oracle-t* runs, each is trained on the same number of tokens that a 220B continual run observes by the end of month t, totaling 1.16T tokens for all seven Oracle models together. Finally, to perform realistic hyperparameter selection, we follow takeaways from (Cha and Cho, 2024) and only use the first 10 timesteps for tuning (see Appx. C for more details).

Evaluation metrics. Each run produces a $T_t \times T_e$ matrix of evaluations E where T_t, T_e are the total number of training/evaluation timesteps, $E_{i,j}$

[2] Here, token counts are given by 20 $\times$ parameters $\times$ Chinchilla multiplier with a $1\times$ multiplier being a near-optimal compute allocation found by Hoffmann et al. (2022).

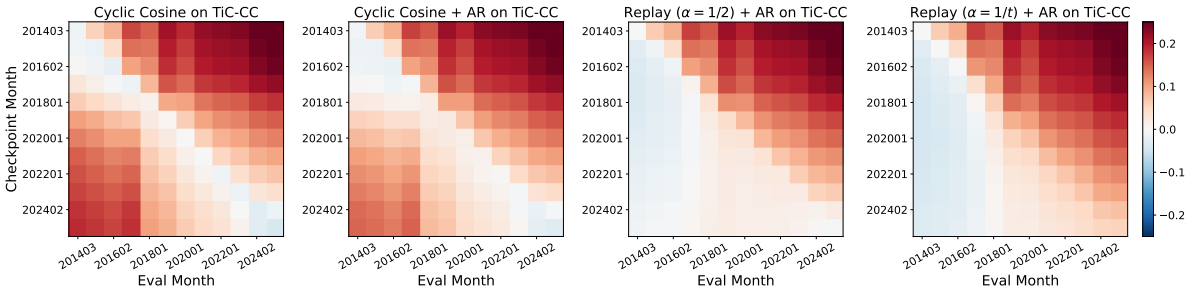


Figure 3: **Continual methods incur different trade-offs between ID and Backward performance on TiC-CC.** We plot selected regret matrices for 3B models and 440B training tokens (lower is better). Overall, Cyclic Cosine leads to strong ID performance (along the diagonal) but also significant forgetting after a few years. This can be partially addressed by using an AR meta-schedule and more significantly by replay. However, too much replay such as with $\alpha_t = 1/t$ scales poorly with a large number of timestamps, significantly sacrificing ID performance.

Table 2: **Loss-based evaluations for various methods.** We report log-perplexity relative to *Oracle-2024-07* models of the same size. While optimizer (top) and regularization-based (bottom) methods trade-off backward transfer with ID performance, replay (middle) is required to obtain the least amount of forgetting. **Bold** values are within one standard deviation of the best in each column for a given token budget, with standard deviations estimated from three runs of Cyclic Cosine. Highlighted values indicate when a continual run outperforms the Oracle series.

Method	Tokens	Params	TiC-CC ↓			TiC-CC-Wiki ↓			TiC-CC-News ↓		
			Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	Bwd.	ID	Fwd.
Cyclic Cosine (std)	220B	1B	0.079 (0.000)	**0.026** (0.000)	**0.154** (0.000)	0.041 (0.000)	0.033 (0.000)	0.072 (0.000)	0.065 (0.000)	0.016 (0.000)	0.105 (0.000)
Cyclic Cosine + AR	220B	1B	0.065	0.040	0.158	0.036	0.033	0.073	0.047	0.017	0.106
Cyclic Rsqrt	220B	1B	0.073	0.031	**0.154**	0.039	0.032	0.071	0.057	**0.015**	0.104
Schedule-Free	220B	1B	0.075	0.036	0.158	0.043	0.037	0.075	0.058	0.019	0.107
Replay ($\alpha = 1/t$)	220B	1B	**0.026**	0.072	0.170	0.024	0.036	0.076	0.008	0.034	0.112
Replay ($\alpha = 1/2$)	220B	1B	0.028	0.042	0.159	0.026	0.032	0.073	0.017	0.019	0.107
Replay ($\alpha = 1/t$) + AR	220B	1B	0.029	0.080	0.173	**0.022**	0.038	0.076	**0.007**	0.038	0.113
Replay ($\alpha = 1/2$) + AR	220B	1B	0.029	0.054	0.163	0.025	0.033	0.073	0.012	0.022	0.107
LwF	220B	1B	0.079	0.027	**0.154**	0.042	0.034	0.073	0.066	0.016	0.105
EWC	220B	1B	0.067	0.035	0.155	0.035	**0.031**	**0.070**	0.050	**0.015**	**0.103**
Oracle Series	1.16T	1B	-0.003	0.035	0.154	0.004	0.016	0.062	-0.008	0.019	0.100
Cyclic Cosine (std)	220B	3B	0.072 (0.000)	**0.027** (0.000)	**0.161** (0.000)	0.038 (0.000)	0.032 (0.000)	0.074 (0.000)	0.058 (0.000)	0.015 (0.000)	0.109 (0.000)
Cyclic Cosine + AR	220B	3B	0.058	0.040	0.166	0.032	0.031	0.074	0.041	0.017	0.110
Cyclic Rsqrt	220B	3B	0.065	0.030	0.162	0.033	0.030	0.073	0.049	0.015	**0.108**
ScheduleFree	220B	3B	0.065	0.036	0.164	0.036	0.033	0.076	0.049	0.017	0.110
Replay ($\alpha = 1/t$)	220B	3B	**0.023**	0.074	0.178	0.020	0.036	0.078	0.005	0.035	0.117
Replay ($\alpha_t = 1/2$)	220B	3B	0.024	0.042	0.167	0.024	0.031	0.074	0.013	0.019	0.111
Replay ($\alpha = 1/t$) + AR	220B	3B	0.026	0.083	0.181	**0.019**	0.037	0.079	**0.004**	0.039	0.119
Replay ($\alpha = 1/2$) + AR	220B	3B	0.025	0.055	0.171	0.022	0.032	0.076	0.009	0.022	0.112
LwF	220B	3B	0.072	**0.027**	**0.161**	0.038	0.032	0.074	0.058	0.015	0.109
EWC	220B	3B	0.061	0.032	0.162	0.031	**0.029**	**0.071**	0.046	**0.014**	**0.108**
Cyclic Cosine (std)	440B	3B	0.082 (0.000)	**-0.011** (0.000)	**0.145** (0.000)	0.029 (0.000)	0.015 (0.000)	0.059 (0.000)	0.071 (0.000)	-0.001 (0.000)	0.099 (0.000)
Cyclic Cosine + AR	440B	3B	0.058	-0.002	0.148	0.014	**0.009**	**0.057**	0.044	**-0.005**	**0.097**
Cyclic Rsqrt	440B	3B	0.067	-0.007	0.146	0.018	0.010	**0.057**	0.055	-0.004	**0.097**
Schedule-Free	440B	3B	0.063	-0.004	0.147	0.017	0.011	0.059	0.049	-0.004	0.098
Replay ($\alpha = 1/t$)	440B	3B	0.001	0.044	0.164	0.003	0.016	0.062	-0.009	0.013	0.105
Replay ($\alpha = 1/2$)	440B	3B	0.007	0.007	0.151	0.010	0.013	0.059	0.005	-0.000	0.099
Replay ($\alpha = 1/t$) + AR	440B	3B	**-0.003**	0.050	0.166	**-0.006**	0.013	0.061	**-0.017**	0.014	0.105
Replay ($\alpha = 1/2$) + AR	440B	3B	-0.002	0.016	0.154	-0.001	**0.009**	**0.057**	-0.009	-0.002	0.098
LwF	440B	3B	0.082	**-0.011**	**0.145**	0.029	0.015	0.059	0.072	-0.001	0.099
EWC	440B	3B	0.055	0.011	0.152	0.017	0.015	0.061	0.041	0.001	0.100
Oracle Series	1.16T	3B	-0.003	0.037	0.163	0.004	0.017	0.066	-0.007	0.020	0.107

is the performance of the model trained after seeing data up to month i and evaluated on the month j. To control for inherent difficulty differences across evaluation months, we measure the *regret* $R_{i,j} = E_{i,j} - E^*_j$ where E^*_j is the performance of *Oracle-2024-07* on month j. Following Garg et al. (2024), we consider the following summary metrics, first defined assuming $T_t = T_e = T$ (deferring the discussion of the $T_t \neq T_e$ case to Appx. B.4):

- In-distribution (ID) performance: averages along the matrix diagonal, i.e., $\sum_{i=1}^{T} = R_{i,i}/T$.
- Backward transfer: averages the lower triangular of R, i.e., $\sum_{i=1}^{T} \sum_{j<i} \frac{R_{i,j}}{T(T-1)/2}$, capturing how well continual checkpoints do on older months.
- Forward transfer: averages the upper triangular of R analogously to backward transfer, capturing how well methods do on unseen future months.

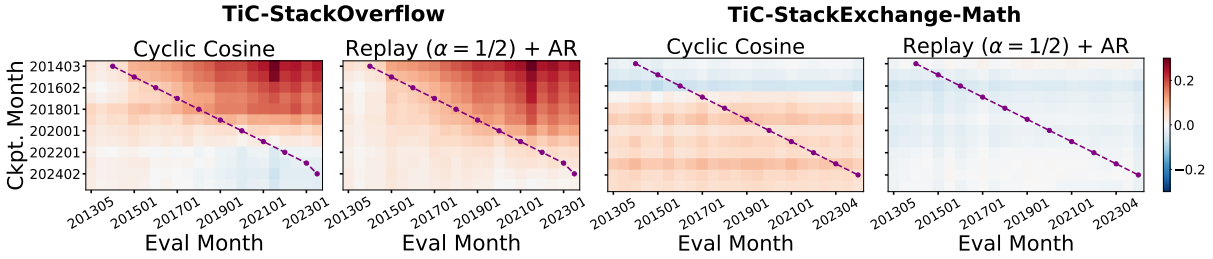


Figure 4: **Replay helps on TIC-STACKE-MATH but hurts on domain that evolve more quickly such as TIC-STACKOVERFLOW.** The purple lines trace out when the training and evaluation timestamps are closest.

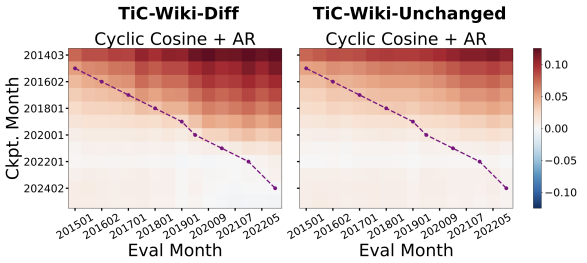


Figure 5: **On TIC-WIKI, performance on an evaluation month can peak years after the corresponding CC dump is seen.** This occurs even without replay.

6.1 Held-out performance on TIC-CC

In Tab. 2 and Fig. 3, we first explore how well continual methods learn and retain various parts of our *general web-scale training distribution*, TIC-CC. Overall, we observe significant trade-offs between ID and backward transfer, with key findings below:

Continual pretraining outperforms the Oracle series with 62% less compute on TIC-CC. Replay (α_t = 1/2) + AR at 440B tokens outperforms the Oracle series on almost all metrics, coming within 0.0001 on backward transfer. From our 220B runs, we see that gaps do remain between (token-matched) continually trained models at timestamp t and Oracle-t. The key though is that by reusing models, continual learning allows for more frequent checkpoints (that have seen more tokens) while maintaining cheaper total costs.

Cyclic Cosine achieves the best ID performance on TIC-CC but also the most forgetting. As shown by Tab. 5 in Appx. C, the best maximum learning rate (LR) per cycle is 1e-4, notably 30× smaller than that used for the initial May-2013 pretraining. This differs from Ibrahim et al. (2024); Gupta et al. (2023) who suggest rewarming up to a similar LR, likely because our continual rounds are smaller and more frequent. Indeed, higher LRs degraded all metrics, while lower LRs improved backward transfer at the cost of ID. Compared to 1e-4, the AR schedule marked the opposite end of the spectrum, offering the best backward transfer compared to setting any fixed maximum LR.

Replay is essential for addressing forgetting. Based on backward transfer and Figs. 16 and 17, all non-replay methods show significant forgetting at later checkpoints. Optimizer tweaks and EWC can somewhat reduce forgetting by sacrificing some ID performance. However, additionally applying replay can further improve backward transfer by 60% for 220B runs, a gap which only *widens* as we scale up to 440B tokens. Between replay variants, $\alpha_t = 1/t$ achieves the lowest forgetting but $\alpha_t = 1/2$ offers a better practical trade-off on TIC-CC, resulting in marginally worse Backward but substantially better ID. This differs from TiC-CLIP's recommendation of $\alpha_t = 1/t$, likely since decreasing the ratio of new data becomes problematic in our setup which uses 10× more timesteps.

Forgetting remedies may not sacrifice ID on specific subsets of web-data. Comparing ID and forward transfer, Wiki appears to evolve more slowly than News, with both changing less rapidly than TIC-CC. Also, as shown in Fig. 6 (Appendix A), the *prevalance* of specific domains can vary over time (in addition to their contents), with sharp drop-offs in 2017 for both subsets. This may explain why at 440B scale, AR schedules and replay can outperform Cyclic Cosine's ID performance on TIC-CC-WIKI and TIC-CC-NEWS, despite being worse for ID on full TIC-CC.

For smaller models, forgetting is more of a challenge but method rankings remain similar. We observe that ID for 1B methods (relative to the 1B Oracle) is quite close to ID for 3B methods (relative to the 3B Oracle). However, Backward metrics are worse at 1B especially when not using replay. This suggests that, perhaps as expected, models struggle more to retain older knowledge when they have fewer parameters. Overall though, we obtain similar conclusions between the two scales in terms of what methods best mitigate forgetting: in-

Table 3: **Benchmarking continual methods on downstream evaluations (3B Models).** For all dynamic evaluations, we report perplexity relative to *Oracle-2024-07* with log-scaling. CORE is an average of the accuracies of 22 downstream zero/few-shot tasks used by Li et al. (2024a), evaluated only on the final model checkpoint (without scaling by an Oracle). **Bold** values are within one standard deviation (estimated with 3 runs of Cyclic Cosine) of the best in each column. Highlighted values indicate matching the Oracle series (within one standard deviation).

Method	Tokens	TiC-Wiki-Diff ↓			TiC-Wiki-Unch. ↓			TiC-StackOverflow ↓			TiC-CD-PyTorch↓		
		Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	Bwd.	ID	Fwd.
Cyclic Cosine (std)	440B	0.018 (0.000)	0.026 (0.000)	0.072 (0.001)	0.026 (0.001)	0.031 (0.000)	0.056 (0.001)	0.017 (0.001)	**0.045** (0.002)	**0.142** (0.002)	-0.020 (0.004)	-0.002 (0.001)	0.175 (0.002)
Cyclic Cosine + AR	440B	**0.010**	**0.023**	**0.071**	0.010	**0.022**	**0.054**	0.026	0.057	**0.142**	0.012	0.027	0.195
Cyclic Rsqrt	440B	0.012	0.024	**0.070**	0.014	0.024	**0.053**	0.024	0.055	0.145	-0.001	0.015	0.187
ScheduleFree	440B	0.012	0.025	0.073	0.014	0.025	0.056	0.038	0.066	**0.144**	0.030	0.044	0.197
Replay ($\alpha = 1/t$)	440B	0.019	0.038	0.078	0.014	0.028	0.057	0.039	0.087	0.176	0.106	0.119	0.238
Replay ($\alpha = 1/2$)	440B	0.015	0.029	0.073	0.017	0.027	0.055	0.027	0.065	0.158	0.020	0.032	0.189
Replay ($\alpha = 1/t$) + AR	440B	0.015	0.036	0.077	**0.005**	0.023	0.056	0.057	0.102	0.179	0.141	0.151	0.261
Replay ($\alpha = 1/2$) + AR	440B	**0.010**	0.027	0.073	0.007	**0.022**	**0.054**	0.036	0.074	0.160	0.058	0.070	0.214
LwF	440B	0.019	0.027	0.072	0.026	0.031	0.056	**0.014**	**0.047**	**0.142**	**-0.028**	**-0.011**	**0.171**
EWC	440B	0.015	0.030	0.074	0.016	0.029	0.058	0.039	0.073	0.155	0.055	0.069	0.211
Oracle Series	1.16T	0.014	0.035	0.080	0.013	0.030	0.061	0.012	0.056	0.146	0.035	0.057	0.196

Method	Tokens	TiC-StackE-Math ↓			TiC-CD-NumPy ↓			Static Evals. ↑
		Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	CORE (DCLM)
Cyclic Cosine	440B	0.048 (0.002)	0.030 (0.002)	0.013 (0.000)	0.054 (0.002)	0.070 (0.001)	0.065 (0.002)	49.6 (0.3)
Cyclic Cosine + AR	440B	0.023	0.008	-0.001	0.044	0.057	0.055	48.8
Cyclic Rsqrt	440B	0.032	0.017	0.006	0.050	0.061	0.060	49.4
ScheduleFree	440B	0.036	0.023	0.010	0.081	0.091	0.072	49.4
Replay ($\alpha = 1/t$)	440B	-0.025	-0.025	**-0.022**	0.035	0.038	**0.041**	49.4
Replay ($\alpha = 1/2$)	440B	0.000	-0.008	-0.011	**0.029**	0.038	0.044	**50.1**
Replay ($\alpha = 1/t$) + AR	440B	**-0.028**	**-0.027**	**-0.022**	0.031	**0.032**	0.047	49.3
Replay ($\alpha = 1/2$) + AR	440B	-0.015	-0.019	-0.017	0.036	0.039	**0.043**	49.3
LwF	440B	0.049	0.031	0.013	0.059	0.077	0.070	49.6
EWC	440B	0.014	0.004	0.000	0.063	0.066	0.059	49.0
Oracle series	1.16T	-0.025	-0.028	-0.022	0.008	0.008	0.015	50.6

creasing model size can slightly improve relative Backward regret but not nearly as much as switching from non-replay to replay methods.

6.2 Downstream evaluations

Here, Tab. 3 and Fig. 4 show results for our downstream evaluations. In contrast to TIC-CC evaluations, LLMs are often evaluated on curated tasks that may not be completely nor temporally aligned with CC dumps as outdated data can exist in newer dumps (Cheng et al., 2024). Hence, we observe that methods exhibit trade-offs *across different evaluations*, highlighting the challenges of performing on all domains while pretraining on general web-data.

Continual methods outperform Oracle retraining on many downstream evaluations. While Replay ($\alpha_t = 1/2$) + AR at 440B tokens outperforms the Oracle series on TIC-CC, the comparison is more nuanced for downstream tasks. Many continual methods show significant gains on TIC-WIKI evaluations, while the Oracle series maintains an edge on TIC-CODEDOCS-NUMPY and TIC-STACKE-English (see Appx. D). Meanwhile, methods like Cyclic Cosine outperform the Oracles on TIC-STACKOVERFLOW and TIC-CODEDOCS-PYTORCH but fall short on TIC-STACKE-MATH where only Replay ($\alpha = 1/t$) can do so.

Replay may underperform on faster-evolving domains but helps when evolution is slower and older dumps contain more relevant data. For some slowly-evolving domains like TIC-STACKE-MATH, earlier CC dumps (pre-Feb-2016) provide the most value, making both replay and AR schedules beneficial (see Fig. 4). In contrast, for TIC-STACKOVERFLOW, larger performance gradients exist across time and using less historical data improves all metrics. A similar pattern is evident in TIC-CODEDOCS, where replay improves all metrics for NumPy (released in 1995) but harms performance for PyTorch (released in 2016). The heatmaps in Appx. D show that Cyclic Cosine models initially improve on NumPy up to 2016 before forgetting it until 2024, suggesting replay is necessary to retain this knowledge (which features most prominently in earlier dumps). Conversely, replay hurts for PyTorch by overemphasizing data that was crawled from before its release.

Forgetting earlier CC dumps may have less impact on general factual knowledge. On TIC-WIKI, replay shows surprisingly different behavior than on TIC-CC-Wiki. Non-replay methods remain competitive not just for ID but also for backward transfer, even on TIC-WIKI-Unchanged. Fig. 5 reveals that TIC-WIKI performance often

peaks years after the corresponding CC dump, even without replay. This suggests two key insights: (1) by focusing on specific segments and proper nouns rather than all tokens, TIC-WIKI may better isolate persistent factual knowledge rather than temporal differences such as formatting changes; (2) knowledge captured in TIC-WIKI can be effectively learned from later CC dumps, possibly due to delayed alignment between CC's Wikipedia crawls and TIC-WIKI's comprehensive coverage.

On static evaluations, continual methods leave room for improvement. As shown in Tab. 3, most continual runs perform similarly on CORE tasks from Li et al. (2024a), with a persistent gap to Oracle-2024-07. The remaining difference could stem from either the Oracle's unrestricted data access or our initialization bias toward May-2013 data. This initialization achieves a score of 48.5 while starting from it and training simultaneously on all 113 remaining months achieves 49.9, landing between our best 220B (Appx. D) and 440B runs.

7 Conclusion

In this work, we introduced a benchmark uniquely scoped for web-scale continual LLM pretraining, consisting of 114 timesteps of Common Crawl training data and several time-stratified evaluations in the form of TIC-WIKI, TIC-STACKE, and TIC-CODEDOCS. Using these assets, we highlighted key observations about balancing forgetting and plasticity on general web-data as well as how this translates to nuanced trade-offs across different evaluation domains. Overall, we view our work as a pivotal step towards understanding how to best continually pretrain LLMs and developing even more efficient continual methods.

Limitations

An important limitation of our work is that we were not able to find a method that outperfroms Oracle re-training on all evaluations. To close these remaining gaps, we hope that future efforts can use our assets to explore more creative ways to balance forgetting and plasticity. Some potentially promising directions could be, but are not limited to: (1) *adaptive* strategies for determining replay ratios; (2) adding timestamps to training documents as in Rosin et al. (2022); Dhingra et al. (2022); (3) time-aware *data-centric* interventions such as deduplicating/filtering replay buffers to keep only the most unique/useful knowledge from older dumps.

Another set of limitations relates to our benchmark design and bringing it closer to current common practices for LLM pre-training. First, we focus on general web-data but LLMs are typically trained on a mix of other separately curated sources (e.g. Wikipedia, ArXiv, etc.). It would thus be interesting to expand our training setup to include time-stratified versions of such sources. Another practical consideration is related to the evolution of *tokenizers* over time. Our benchmark fixes the tokenizer across all timesteps and methods but it would be interesting to explore whether it would be beneficial to co-adapt models and tokenizers as language changes over time. Finally, our current dynamic evaluations were limited to perplexity-based metrics. Given the scale of our training runs, we had found that it was difficult to find accuracy-based variants of our evaluations that resulted in meaningful signal compared to noise.

Ethical Considerations

This paper presents work whose goal is to advance the efficiency of training and updating large language models. These improvements could help democratize LM development by enabling resource-constrained organizations to participate in research and development, while reducing the environmental footprint of model training through decreased energy consumption. However, increased accessibility of LM training also increases any risks that pertain to language models as a technology, such as the potential for malicious actors to spread disinformation.

Since our training data comes from Common Crawl and the general web, there may exist offensive and personally identifying content. This risk carries over from the original datasets we built ours from (Common Crawl, DCLM-Pool). Removing such data may affect the performance of models. Future work exploring the filtering of the original datasets may be directly applied to our training and evaluations as well.

Acknowledgements

We are highly grateful to the many developers of the OpenLM library which was used extensively in this work. In particular, we'd like to thank George Smyrnis, Achal Dave, and Samir Yitzhak Gadre for helpful discussions regarding its usage. We'd also like to thank Chun-Liang Li, Pavan Kumar Anasosalu Vasu, Preetum Nakkiran, and Russ Webb for helping review earlier versions of the manuscript.

References

Oshin Agarwal and Ani Nenkova. 2022. Temporal effects on pre-trained models for language processing tasks. *Transactions of the Association for Computational Linguistics*, 10:904–921.

Jason Ansel, Edward Z. Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. 2024. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, pages 929–947. ACM.

Yogesh Balaji, Mehrdad Farajtabar, Dong Yin, Alex Mott, and Ang Li. 2020. The effectiveness of memory replay in large scale continual learning. *arXiv preprint arXiv:2010.02418*.

Loïc Barrault, Magdalena Biesialska, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Yvette Graham, Roman Grundkiewicz, Barry Haddow, Matthias Huck, Eric Joanis, Tom Kocmi, Philipp Koehn, Chi-kiu Lo, Nikola Ljubešić, Christof Monz, Makoto Morishita, Masaaki Nagata, Toshiaki Nakazawa, Santanu Pal, Matt Post, and Marcos Zampieri. 2020. Findings of the 2020 conference on machine translation (WMT20). In *Proceedings of the Fifth Conference on Machine Translation*, pages 1–55, Online. Association for Computational Linguistics.

Himanshu Beniwal, Mayank Singh, et al. 2024. Remember this event that year? assessing temporal information and reasoning in large language models. *arXiv preprint arXiv:2402.11997*.

Janek Bevendorff, Martin Potthast, and Benno Stein. 2021. FastWARC: Optimizing Large-Scale Web Archive Analytics. In *3rd International Symposium on Open Search Technology (OSSYM 2021)*. International Open Search Symposium.

Janek Bevendorff, Benno Stein, Matthias Hagen, and Martin Potthast. 2018. Elastic ChatNoir: Search Engine for the ClueWeb and the Common Crawl. In *Advances in Information Retrieval. 40th European Conference on IR Research (ECIR 2018)*, Lecture Notes in Computer Science, Berlin Heidelberg New York. Springer.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of the ACL Workshop on Challenges & Perspectives in Creating Large Language Models*.

Sungmin Cha and Kyunghyun Cho. 2024. Hyperparameters in continual learning: A reality check. *arXiv preprint arXiv:2403.09066*.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2018. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*.

Jie Chen, Zhipeng Chen, Jiapeng Wang, Kun Zhou, Yutao Zhu, Jinhao Jiang, Yingqian Min, Wayne Xin Zhao, Zhicheng Dou, Jiaxin Mao, Yankai Lin, Ruihua Song, Jun Xu, Xu Chen, Rui Yan, Zhewei Wei, Di Hu, Wenbing Huang, and Ji-Rong Wen. 2024. Towards effective and efficient continual pretraining of large language models. *arXiv preprint arXiv:2407.18743*.

Jeffrey Cheng, Marc Marone, Orion Weller, Dawn Lawrie, Daniel Khashabi, and Benjamin Van Durme. 2024. Dated data: Tracing knowledge cutoffs in large language models. In *First Conference on Language Modeling*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint*. https://arxiv.org/abs/1803.05457.

Aaron Defazio, Xingyu, Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. 2024. The road less scheduled. *arXiv preprint arXiv: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. 2024. Language modeling is compression. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Bhuwan Dhingra, Jeremy R Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W Cohen. 2022. Time-aware language models as temporal knowledge bases. *Transactions of the Association for Computational Linguistics*, 10:257–273.

Felix Drinkall, Eghbal Rahimikia, Janet B. Pierrehumbert, and Stefan Zohren. 2024. Time machine GPT. In *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 3281–3292. Association for Computational Linguistics.

Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. 2020. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3762–3773. PMLR.

Bahare Fatemi, Mehran Kazemi, Anton Tsitsulin, Karishma Malkan, Jinyeong Yim, John Palowitch, Sungyong Seo, Jonathan Halcrow, and Bryan Perozzi. 2024. Test of time: A benchmark for evaluating llms on temporal reasoning. *arXiv preprint arXiv:2406.09170*.

Saurabh Garg, Mehrdad Farajtabar, Hadi Pouransari, Raviteja Vemulapalli, Sachin Mehta, Oncel Tuzel, Vaishaal Shankar, and Fartash Faghri. 2024. Tic-clip: Continual training of clip models. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Kshitij Gupta, Benjamin Thérien, Adam Ibrahim, Mats Leon Richter, Quentin Gregory Anthony, Eugene Belilovsky, Irina Rish, and Timothée Lesort. 2023. Continual pre-training of large language models: How to re-warm your model? In *Workshop on Efficient Systems for Foundation Models @ ICML2023*.

Wes Gurnee and Max Tegmark. 2024. Language models represent space and time. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online. Association for Computational Linguistics.

Suchin Gururangan, Mitchell Wortsman, Samir Yitzhak Gadre, Achal Dave, Maciej Kilian, Weijia Shi, Jean Mercat, Georgios Smyrnis, Gabriel Ilharco, Matt Jordan, Reinhard Heckel, Alex Dimakis, Ali Farhadi, Vaishaal Shankar, and Ludwig Schmidt. 2023. open_lm: a minimal but performative language modeling (lm) repository. GitHub repository.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature*, 585(7825):357–362.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2302.00487*.

Matthew Honnibal and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 7(1):411–420.

Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. 2018. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*.

Adam Ibrahim, Benjamin Thérien, Kshitij Gupta, Mats L. Richter, Quentin Gregory Anthony, Eugene Belilovsky, Timothée Lesort, and Irina Rish. 2024. Simple and scalable strategies to continually pre-train large language models. *Trans. Mach. Learn. Res.*, 2024.

Joel Jang, Seonghyeon Ye, Changho Lee, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, and Minjoon Seo. 2022a. Temporalwiki: A lifelong benchmark for training and evaluating ever-evolving language models. *arXiv preprint arXiv:2204.14211*.

Joel Jang, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, Stanley Jungkyu Choi, and Minjoon Seo. 2022b. Towards continual knowledge learning of language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Zhen Jia, Philipp Christmann, and Gerhard Weikum. 2024. Tiq: A benchmark for temporal question answering with implicit time constraints. In *Companion Proceedings of the ACM on Web Conference 2024*, pages 1394–1399.

Xisen Jin, Dejiao Zhang, Henghui Zhu, Wei Xiao, Shang-Wen Li, Xiaokai Wei, Andrew O. Arnold, and Xiang Ren. 2022. Lifelong pretraining: Continually adapting language models to emerging corpora. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 4764–4780. Association for Computational Linguistics.

Jungo Kasai, Keisuke Sakaguchi, Ronan Le Bras, Akari Asai, Xinyan Yu, Dragomir Radev, Noah A Smith, Yejin Choi, Kentaro Inui, et al. 2024. Realtime qa: what's the answer right now? *Advances in Neural Information Processing Systems*, 36.

Yujin Kim, Jaehong Yoon, Seonghyeon Ye, Sangmin Bae, Namgyu Ho, Sung Ju Hwang, and Se-Young Yun. 2024. Carpe diem: On the evaluation of world knowledge in lifelong language models. In *Proceedings of the 2024 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024, pages 5401–5415. Association for Computational Linguistics.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Angeliki Lazaridou, Adhi Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Tomas Kocisky, Sebastian Ruder, et al. 2021. Mind the gap: Assessing temporal generalization in neural language models. *Advances in Neural Information Processing Systems*, 34:29348–29363.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, et al. 2024a. Datacomp-lm: In search of the next generation of training sets for language models. *arXiv preprint arXiv:2406.11794*.

Yucheng Li, Yunhao Guo, Frank Guerin, and Chenghua Lin. 2024b. Evaluating large language models for generalization and robustness via data compression. *arXiv preprint arXiv:2402.00861*.

Zhizhong Li and Derek Hoiem. 2018. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947.

Adam Liška, Tomáš Kočiský, Elena Gribovskaya, Tayfun Terzi, Eren Sezener, Devang Agrawal, Cyprien de Masson d'Autume, Tim Scholtes, Manzil Zaheer, Susannah Young, Ellen Gilsenan-McMahon Sophia Austin, Phil Blunsom, and Angeliki Lazaridou. 2022. Streamingqa: A benchmark for adaptation to new knowledge over time in question answering models. *arXiv preprint arXiv:2205.11388*.

Vincenzo Lomonaco, Lorenzo Pellegrini, Pau Rodriguez, Massimo Caccia, Qi She, Yu Chen, Quentin Jodelet, Ruiping Wang, Zheda Mai, David Vazquez, et al. 2022. Cvpr 2020 continual learning in computer vision competition: Approaches, results, current challenges and future directions. *Artificial Intelligence*, 303:103635.

David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30.

Daniel Loureiro, Francesco Barbieri, Leonardo Neves, Luis Espinosa Anke, and Jose Camacho-collados. 2022. TimeLMs: Diachronic language models from Twitter. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 251–260, Dublin, Ireland. Association for Computational Linguistics.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*.

Kelvin Luu, Daniel Khashabi, Suchin Gururangan, Karishma Mandyam, and Noah A. Smith. 2022. Time waits for no one! analysis and challenges of temporal misalignment. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 5944–5958. Association for Computational Linguistics.

Sanket Vaibhav Mehta, Darshan Patil, Sarath Chandar, and Emma Strubell. 2023. An empirical investigation of the role of pre-training in lifelong learning. *Journal of Machine Learning Research*, 24(214):1–50.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, and Hassan Ghasemzadeh. 2020a. Dropout as an implicit gating mechanism for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 232–233.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. 2020b. Understanding the role of training regimes in continual learning. *Advances in Neural Information Processing Systems*, 33:7308–7320.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. 2022a. Fast model editing at scale. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D Manning, and Chelsea Finn. 2022b. Memory-based model editing at scale. In *International Conference on Machine Learning*, pages 15817–15831. PMLR.

MosaicML. 2023. llm-foundry/scripts/eval/local_data/EVAL_GAUNTLET.md at main · mosaicml/llm-foundry — github.com. https://github.com/mosaicml/llm-foundry/blob/main/scripts/eval/local_data/EVAL_GAUNTLET.md.

Ali Mousavi, Xin Zhan, He Bai, Peng Shi, Theodoros Rekatsinas, Benjamin Han, Yunyao Li, Jeffrey Pound, Joshua M. Susskind, Natalie Schluter, Ihab F. Ilyas, and Navdeep Jaitly. 2024a. Construction of paired knowledge graph - text datasets informed by cyclic evaluation. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pages 3782–3803. ELRA and ICCL.

Seyed Mahed Mousavi, Simone Alghisi, and Giuseppe Riccardi. 2024b. Is your llm outdated? benchmarking llms & alignment algorithms for time-sensitive knowledge. *arXiv preprint arXiv:2404.08700*.

Kai Nylund, Suchin Gururangan, and Noah A. Smith. 2024. Time is encoded in the weights of finetuned language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 2571–2587. Association for Computational Linguistics.

Jupinder Parmar, Sanjev Satheesh, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. 2024. Reuse, don't retrain: A recipe for continued pretraining of language models. *arXiv preprint arXiv:2407.07263*.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *arXiv preprint*. https://arxiv.org/abs/2306.01116.

Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. 2020. Gdumb: A simple approach that questions our progress in continual learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 524–540. Springer.

Yujia Qin, Jiajie Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2022. ELLE: Efficient lifelong pre-training for emerging data. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2789–2810, Dublin, Ireland. Association for Computational Linguistics.

Anastasia Razdaibiedina, Yuning Mao, Rui Hou, Madian Khabsa, Mike Lewis, and Amjad Almahairi. 2023. Progressive prompts: Continual learning for language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010.

Guy D Rosin, Ido Guy, and Kira Radinsky. 2022. Time masking for temporal language models. In *Proceedings of the fifteenth ACM international conference on Web search and data mining*, pages 833–841.

Karsten Roth, Vishaal Udandarao, Sebastian Dziadzio, Ameya Prabhu, Mehdi Cherti, Oriol Vinyals, Olivier Hénaff, Samuel Albanie, Matthias Bethge, and Zeynep Akata. 2024. A practitioner's guide to continual multimodal pretraining. *arXiv preprint arXiv:2408.14471*.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. 2018. Progress & compress: A scalable framework for continual learning. In *International conference on machine learning*, pages 4528–4537. PMLR.

Gido M Van de Ven and Andreas S Tolias. 2019. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*.

Tu Vu, Mohit Iyyer, Xuezhi Wang, Noah Constant, Jerry W. Wei, Jason Wei, Chris Tar, Yun-Hsuan Sung, Denny Zhou, Quoc V. Le, and Thang Luong. 2024. Freshllms: Refreshing large language models with search engine augmentation. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 13697–13720. Association for Computational Linguistics.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. 2024. A comprehensive survey of continual learning: Theory, method and application.

Tongtong Wu, Linhao Luo, Yuan-Fang Li, Shirui Pan, Thuy-Trang Vu, and Gholamreza Haffari. 2024. Continual learning for large language models: A survey.

Klim Zaporojets, Lucie-Aimée Kaffee, Johannes Deleu, Thomas Demeester, Chris Develder, and Isabelle Augenstein. 2022. Tempel: Linking dynamically evolving and newly emerging entities. *Advances in Neural Information Processing Systems*, 35:1850–1866.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Annual Meeting of the Association for Computational Linguistics (ACL)*. https://aclanthology.org/P19-1472.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. 2022. Scaling vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12104–12113.

Bowen Zhao, Zander Brumbaugh, Yizhong Wang, Hannaneh Hajishirzi, and Noah A. Smith. 2024. Set the clock: Temporal alignment of pretrained language models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 15015–15040. Association for Computational Linguistics.

Da-Wei Zhou, Fu-Yun Wang, Han-Jia Ye, and De-Chuan Zhan. 2023. Pycil: A python toolbox for class-incremental learning.

A Dataset Construction

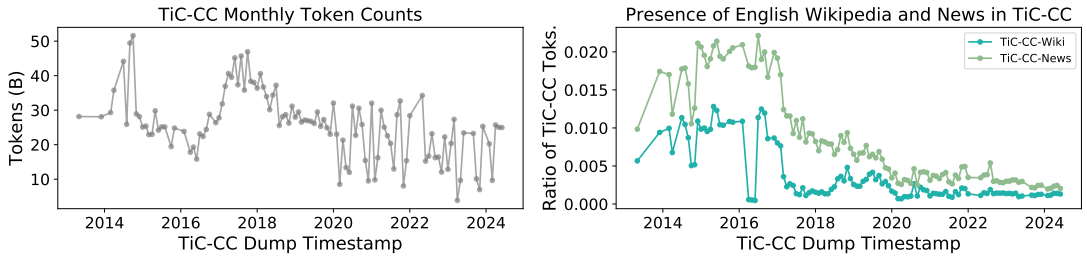


Figure 6: We plot the total number of tokens per month in TIC-CC (left) as well as the proportion of those tokens coming from our TIC-CC-WIKI and TIC-CC-NEWS subsets (right).

We build upon the existing pipeline and assets from DataComp-LM (Li et al., 2024a) to build our dataset, only altering steps that rely on global operations across months.

Initial pool and temporal splitting. We start with DCLM-Pool (Li et al., 2024a) which is available by CC-BY-4 license and contains all CC dumps between May-2013 and December-2022. The only pre-processing that has been done on this pool is to parse the HTML (contained in WARC files of CC) into plaintext for each webpage via the open-source `resiliparse` library (Bevendorff et al., 2018, 2021) with the `main_content` flag set to `True` [3]. In DCLM-Pool, documents are naturally grouped together into files based upon the CC dump, which is indicated by the file prefix. To split the data by month, we simply group files that share the same prefix. Since DCLM-Pool contains data up to December-2022, we also follow their exact download and extraction scripts to obtain more recent data until July-2024.

Data preprocessing and tokenization. Next, we follow DCLM-Baseline's filtering procedure which starts with their implementation of heuristic filters from RefinedWeb (Penedo et al., 2023). We apply these filters, which includes English filtering, independently on each page with no change. However, we have to modify deduplication that removes nearly identical lines and pages. Instead of applying deduplication globally across months as in DCLM-Baseline, we apply the same deduplication method *only within* each month. Finally, we also skip the final classifier-based filtering in DCLM-Baseline, as their classifier was trained on data that comes from all months, including examples generated by recent LLMs such as GPT-4.

Data sampling and held-out sets. DCLM-Pool was partitioned randomly into 10 equally sized "global shards". Within our monthly splits, we also maintain the original global shard assignments. For our training scales, using just one of these global shards within each month is sufficient. Notably though, when we construct evaluation sets such as in (Sec. 4.1), we make sure to sample from a different global shard than the one used for training. This ensures the evaluation data is a sampled from the same distribution as the training data while also being *mostly* held out. Notably, since we do not deduplicate across globals shards or months, there could be overlap between training and eval sets across months. However, we observe from Fig. 7 that potential data leakages are unlikely significantly change relative losses values (compared to the Oracle). For each validation set, we cap the maximum number of tokens to 16.7M which corresponds to 8192 sequences for our context length of 2048. For some months of TIC-CC-WIKI and TIC-CC-NEWS, we end up with less than this amount, but the smallest are 5M and 12M respectively.

[3] We use `readability` for parsing code documentations in our TIC-CODEDOCS (https://github.com/mozilla/readability).

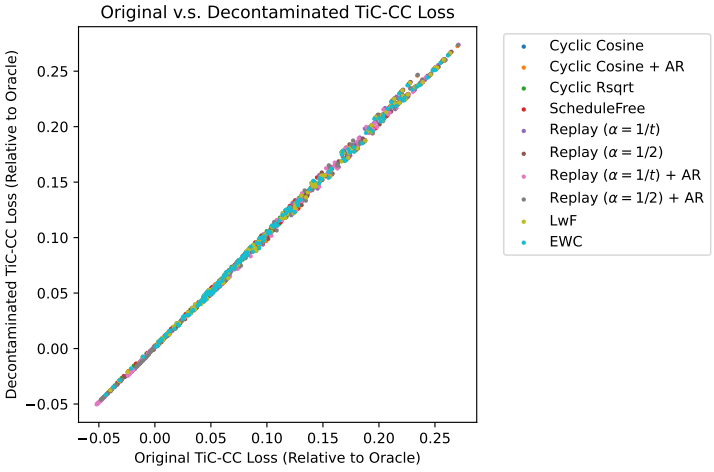


Figure 7: **Findings from TIC-CC are robust to potential data leakages.** We create a decontaminated version of our TIC-CC loss-based evaluation by deduplicating each month's evaluation set using a Bloom Filter pre-populated by the corresponding training set. Overall, across all the methods, checkpoints, and evaluation months we observe strong correlations between using the pre-decontamination (x-axis) and post-decontamination (y-axis) losses (relative to the Oracle).

B Details of Evaluations

B.1 TIC-WIKI

We construct TIC-WIKI from English Wikipedia (available under CC-BY-SA 4.0 license) and Wikidata (available under CC0 license) which are sister projects from the non-profit Wikimedia Foundation. Wikidata is a structured knowledge graph that stores the structured data of Wikipedia and other sister projects. Data on Wikidata is represented in the form of statements in the form of property-value about an item in the simplest form. For example, "Mount Everest is the highest mountain in the world" is represented as Earth (Q2) (item) → highest point (P610) (property) → Mount Everest (Q513) (value) [4]. The triplet (item, property, value) can also be referred to as (subject, relation, object).

TemporalWiki dataset generation. TemporalWiki (Jang et al., 2022a) constructs two forms of evaluations from monthly snapshots of English Wikipedia and Wikidata, which they refer to as intrinsic and extrinsic evaluations. Overall, they are created through the following steps:

1. Generating TWiki-Diffsets by identifying changes and additions between consecutive snapshots of Wikipedia. For new articles, the entire article is added to the Diffset while for existing articles, only the changed or new paragraphs are added. In their study, measuring perplexity on proper nouns separately from TWiki-Diffsets and Unchanged Wikipedia pages serves as intrinsic evaluations (named as such since their study uses these datasets also as their training distributions).
2. Constructing TWiki-Probes by processing two consecutive snapshots of Wikidata. Statements are categorized into changed if the property/value has changed or categorized into unchanged otherwise. In their extrinsic evaluation, perplexity is measured on concatenated versions of the (subject, relation, object) triplets (using space delimiting).
    - Aligning TWiki-Probes with TWiki-Diffsets to ensure changed statements in Twiki-Probes exist in TWiki-Diffsets and unchanged statements exist in unchanged parts of Wikipedia.
    - Further heuristic filtering of TWiki-Probes by removing statements where the subject or object is a substring of the other or the object is more than 5 words. Moreover, a single subject is limited to maximum 1% and relation/object is limited to maximum 5% of the total statements.

In TIC-WIKI we extend and modify TemporalWiki as follows:

1. We expand the timespan from four months to a decade (2014-2024), thus capturing a broader spectrum of knowledge evolution.
2. We construct both a perplexity-based evaluation and a QA-based evaluation.
    - Our perplexity-based evaluation follows their intrinsic evaluation protocol. In our case, this evaluation is no longer fully intrinsic since although some Wikipedia pages exist in Common Crawl, we do not directly train on whole Twiki-Diffsets or Wikipedia dumps at each timestep. Given this, we also chose to carry out our perplexity evaluation without aligning to Wikidata, since Wikidata's knowledge graph does not have full coverage of Wikipedia (Mousavi et al., 2024a) and the temporal alignment between their dumps is not always perfect.
    - In place of their extrinsic evaluation, we create natural question-answer (QA) evaluations with an improved matching process of Wikipedia and Wikidata dumps, and enhancing the robustness of data parsing to format changes over time.

For our perplexity evaluations, our TIC-WIKI evaluation consists of 61 months each with 10k changed and 10k unchanged Wikipedia sentences/paragraphs. To illustrate the characteristics of our generated dataset, we present key statistics in the following figures. Figure 9 shows the number of Wikipedia pages with significant changes between consecutive database dumps over time. This graph provides insight into the volume and temporal distribution of our data generation process, highlighting periods of higher and lower content modification as well as distribution of our dumps.

[4] https://www.wikidata.org/wiki/Help:About_data

For QA evaluations, our models trained in Sec. 6 overall achieved fairly low performance and thus continual trends tended to be noisy. Hence, we focused on the perplexity evaluations in this work. Nevertheless, we discuss our construction for QA examples and release code for generating question-answers as we believe it may be helpful for future studies.

B.1.1 Data Download

Archive dumps. Wikimedia releases regular dumps [5,6], but only retains data for the most recent 4 months. To access historical data, we utilized the Internet Archive [7]. The earliest available dump dates back to November 2014. It is important to note that the archived dumps do not cover every month, with several months missing from the record. In our study, we made use of all available monthly dumps. The filenames of the dumps include the specific date of month that has been collected on, which is typically the 1st or 20th of the month, though this can vary. We include only one dump per month if multiple dumps are available. We check for the first date if not available look for 20th and if neither we start from beginning the month and check for the first available date in that month.

Wikipedia historical dumps. It is possible to reconstruct each version of Wikiepdia using the large history files Wikipeida provide [8]. There are more than 200 historical dumps of English Wikipedia, each sized more than 2GB. Combined together, these files include all revisions and all pages of Wikipeida.

For Wikidata, Wikimedia does not provide historical diff files as Wikipedia except for the last three months [9]. Wikidata file names are formatted similar to `wikidatawiki-20190101-pages-articles.xml.bz2` and available at URLs similar to `https://dumps.wikimedia.org/wikidatawiki/20240401/`.

Each Wikidata dump is approximately 140GB whereas each Wikipeida dump is less than 22GB. Therefore, it is possible to make a version of Wikipedia that keeps track of all changes which results in 200 files of 2GB. But as far as we know there are no such files for Wikidata.

Using the dumps from `archive.org` has several advantages:

- We make sure that we do not leak information from previous timesteps.
- There exists a Wikidata dump close to each Wikipedia dump to be aligned.
- We can use Wiki-Extractor for filtering and remove Wikipeida editorial discussions.

B.1.2 Data preprocessing for perplexity evaluations

We construct perplexity evaluations based on raw Wikipedia diffsets. For these evaluations, we do not rely on an alignment with Wikidata which increases the diversity of TIC-WIKI evaluation with a simpler construction.

Data cleanup. We utilize WikiExtractor [10] to clean up the Wikipedia data. This step extracts the main content and removes extraneous and non-essential characters.

Wikipedia diffsets. To construct consecutive diffs of Wikipedia, we developed a method comparing snapshots of articles from consecutive dumps. For comparing two snapshots of an article, we first remove extraneous whitespace and standardize formatting by preprocessing the text. This involves removing empty lines, stripping newline characters, and creating a normalized version of each line where punctuation is removed and text is converted to lowercase.

Afterward, we use a two-level comparison: first at the paragraph level, then at the sentence level for changed paragraphs. We utilize Python's `difflib.SequenceMatcher` to compare the normalized versions of paragraphs and sentences. This hierarchical method, coupled with normalization, captures substantial edits while filtering out minor or stylistic changes.

We extract and store both changed and unchanged content separately. Changed content includes replaced paragraphs with modified sentences and newly inserted paragraphs. Unchanged content preserves

[5] `https://dumps.wikimedia.org/wikidatawiki/`
[6] `https://dumps.wikimedia.org/enwiki/`
[7] `https://archive.org`
[8] `https://dumps.wikimedia.org/enwiki/latest/` file names containing `pages-meta-history`.
[9] `https://dumps.wikimedia.org/wikidatawiki/`
[10] `https://github.com/attardi/wikiextractor`

paragraphs and sentences that remain identical between versions. New articles are treated as entirely changed content. This approach allows us to focus on meaningful content changes while maintaining the context of unchanged information, providing a comprehensive view of how Wikipedia articles evolve over time. Algorithms 1 and 2 describe the process of constructing Wikipedia diffs and changed/unchanged content.

Evaluation. We create perplexity evaluations from sentences/paragraphs in the changed/unchanged Wikipedia diffs by implementing them in the format of LLM-Foundry (MosaicML, 2023). We also add custom evaluation code to be able to evalute only on proper nouns.

B.1.3 Data preprocessing for QA evaluations

We further construct QA evaluations by extending the alignment with Wikidata. In TemporalWiki, the evaluation was constructed from triplets (subject, relation, object) by concatenating the triplet and measuring perplexity. However, this format will often not result in natural/proper sentences (e.g., missing a verb). For example, we can have a concatenated triplet such as "Florida State Seminoles women's basketball instance of basketball team". As such, a model trained only on natural language may assign an overall lower probability to the object. We construct question-answer pairs described in natural language. Note that the results presented in Sec. 6 are limited to our perplexity evaluations only but we release code for generating QA evaluations as well.

Wikidata diffsets. Next, we extract changed and unchanged Wikidata statements of the form (subject, relation, object) from each consecutive dump. Identical triplets in both dumps are marked as unchanged. Triplets in the new dump not present in the old are categorized as new, with the exception that if a subject entity has more than 10 triplets, the algorithm randomly samples 10 to represent it. When a triplet has the same subject and relation as one in the old dump but a different object and the old and new objects differ only in case (upper/lowercase), the triplet is classified as unchanged; otherwise, it is categorized as new. Triplets from the old dump not found in the new one are implicitly considered removed, but importantly, these are not included in the output sets of changed or unchanged triplets. Throughout this process, the algorithm filters out triplets with overly long object values (more than 5 words) and ensures no duplicates are added. This approach efficiently tracks Wikidata evolution, capturing nuanced changes while managing the volume of data for new entities.

Algorithm 3 describes the process of triplet extraction. In TemporalWiki, a different set of hard-coded rules are used to extract the triplets from Wikidata which perform unreliably for older data. Our approach instead systematically parses Wikidata dumps, which ensures greater efficiency and robustness. TemporalWiki also use web API requests to link Wikipedia pages with Wikidata IDs. However, in our approach we eliminate reliance on web API requests, our method significantly reduces processing time and avoids potential API limitations or downtime.

QA construction. We constructed question-answer pairs in natural language by utilizing Wikipedia sentences. For each triplet, we find a sentence in the Wikipedia page of the subject that mentions both the subject and the object. Then we replace the object with blank and add the additional prefix of "Question: Fill in the blank:" and the suffix of "Answer:". This process results in a natural fill-in-the-blank question for each triplet.

Evaluation. We implement QA evaluations again using LLM-Foundry (MosaicML, 2023), using their `InContextLearningGenerationExactMatchAccuracy` evaluation metric. Each QA sample is contains a context, answer, and aliases. The following is an example QA:

```
{
    "context":
        "Question: Fill in the blank: "
        "Douglas Adams' most notable work is _______."
        "Answer:",
    "answer": "The Hitchhiker's Guide to the Galaxy",
    "aliases": ["A list of possible aliases for the answer"],
}
```

Figure 8: Example of TIC-WIKI for QA evaluations.

Algorithm 1 Construct Wikipedia Consecutive Diffs

```
Input: oldSnapshot, newSnapshot
Output: changedContent, unchangedContent
oldArticles ← ReadArticles(oldSnapshot)
newArticles ← ReadArticles(newSnapshot)
changedContent ← ∅, unchangedContent← ∅
for each articleId in newArticles.keys do
    if articleId in oldArticles then
        oldText ← NormalizeText(oldArticles[articleId].text)
        newText ← NormalizeText(newArticles[articleId].text)
        changed ← ExtractChangedContent(oldText, newText)
        unchanged ← ExtractUnchangedContent(oldText, newText)
        Add (articleId, changed) to changedContent
        Add (articleId, unchanged) to unchangedContent
    else
        Add (articleId, newArticles[articleId].text) to changedContent
    end if
end for
return changedContent, unchangedContent
```

Algorithm 2 Extract Changed Content

```
Input: oldText, newText
Output: changedContent
oldParagraphs ← SplitIntoParagraphs(oldText)
newParagraphs ← SplitIntoParagraphs(newText)
changedContent ← ∅
for each (oldPara, newPara) in Zip(oldParagraphs, newParagraphs) do
    if IsDifferent(oldPara, newPara) then
        oldSentences ← SplitIntoSentences(oldPara)
        newSentences ← SplitIntoSentences(newPara)
        for each (oldSent, newSent) in Zip(oldSentences, newSentences) do
            if IsDifferent(oldSent, newSent) then
                Add newSent to changedContent
            end if
        end for
    end if
end for
return changedContent
```

Algorithm 3 Wikidata Triplet Extraction and Categorization

Require: oldDump, newDump
Ensure: unchanged, new
  unchanged ← {}
  new ← {}
  newEntities ← {}
  for all triplet ∈ newDump **do**
    if triplet ∈ oldDump **then**
      Add triplet to unchanged
    else if hasSameSubjectPredicate(triplet, oldDump) **then**
      oldObject ← getObject(triplet.subject, triplet.predicate, oldDump)
      if equalsIgnoreCase(triplet.object, oldObject) **then**
        Add triplet to unchanged
      else
        Add triplet to new
      end if
    else
      if triplet.subject ∉ oldDump **then**
        Add triplet to newEntities[triplet.subject]
      else
        Add triplet to new
      end if
    end if
  end for
  sampleNewEntityTriplets(newEntities, new)
  filterLongObjects(unchanged, new)
  removeDuplicates(unchanged, new)
  return unchanged, new

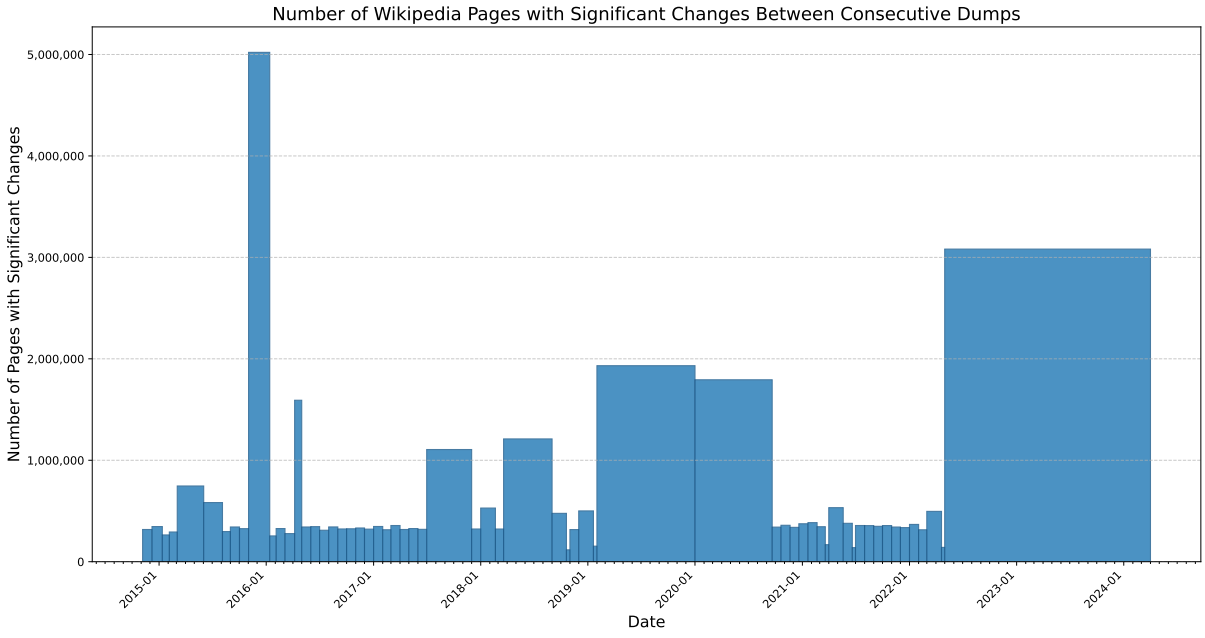


Figure 9: Number of Wikipedia pages with significant Changes between consecutive `archive.org` dumps.

B.2 TIC-STACKE

B.2.1 Data preprocessing

TIC-STACKE spans data from July 2008 through April 2024. The data was sourced from archive.org using the April 2024 dump of StackExchange (available by CC-BY-SA 4.0). Each category in the dump comes with two key files: `Post.xml` and `PostHistory.xml`. `Post.xml` contains information on how answers and questions relate to each other and includes the latest text for each post entry. `PostHistory.xml` records the changes to each post, whether it is a question or an answer.

To construct our dataset, we first build the graph of question-answer relationships based on the `Post.xml`. We then use `PostHistory.xml` to reconstruct exact snapshots of posts at specific timestamps. This allowed us to capture the state of each post at the end of each month, ensuring our data reflected the actual content available at those points in time. We pick out the answers that were accepted by the user who posted the original question and which received at least 4 $\times$ the number of up-votes compared to another answer.

We use the StackOverflow, Mathematics, and English Language & Usage categories in this work but our code can also be used to process any additional categories from the overall 182 categories. Some categories had insufficient questions in a single month to provide statistically significant results. In such cases, we combined data from consecutive months, ensuring that each time frame contains at least 500 questions.

Given the question title and body as the query, we measure answer-perplexity ($\text{ppl}_{\text{answer}}$) on high-quality selected answers. Here is an example QA for TIC-STACKE:

```
{
    "query":
        "Question Title: If squaring a number means multiplying that number"
        " with itself then shouldn't taking square root of a number mean "
        " to divide a number by itself?"
        "\n\n"
        "Question Body: If squaring a number means multiplying that number"
        " with itself then shouldn't taking square root of a number mean"
        " to divide a number by itself?"
        "\n\n"
        "For example the square of $2$  is $2^2=2 \\cdot 2=4 $ ."
        "\n\n"
        "But square root of $2$ is not $\\frac{2}{2}=1$ .",
    "answer":
        "`taking square root` means _reversing_ the effect of `squaring`."
        " Dividing a number by itself does not do that"
        " (but rather always returns 1 as you noted)."
        "\n\n"
        "Compare your question to: if doubling a number means adding it to "
        "itself, shouldn't halving a number mean subtracting it from itself?"
        "Answer: obviously not."
}
```

Figure 10: Example of TIC-STACKE question and answers for Math.

B.2.2 Analysis of StackExchange Data

This section presents additional analysis of question-answer patterns across the top 20 categories of StackExchange, with a focus on StackOverflow, Mathematics, and English Language & Usage. Figure 11 shows the distribution of questions across the top 20 StackExchange categories. Then for our three chosen categories, Figure 12 shows the number of questions asked per month, Figure 14 presents the distribution of answer counts per question, and Figure 13 illustrates the distribution of question lengths.

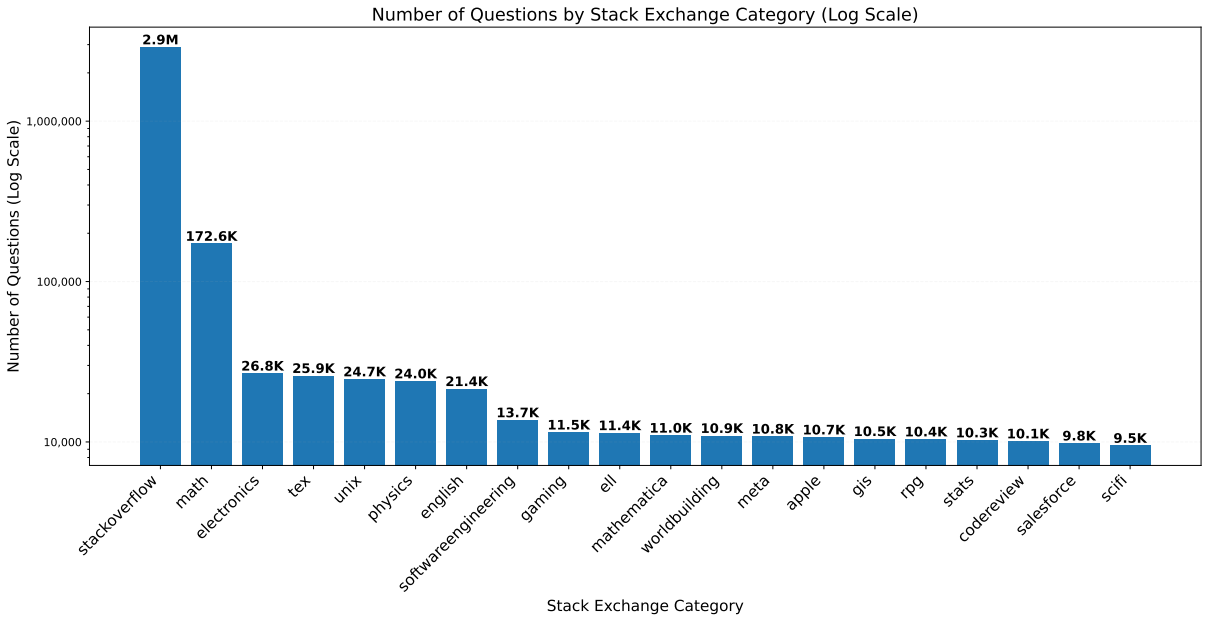
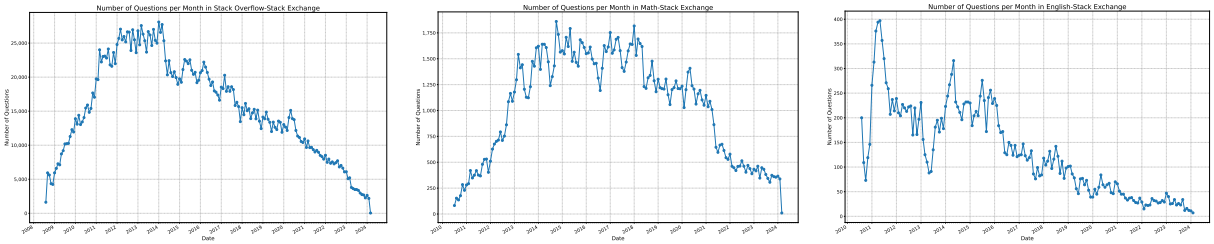
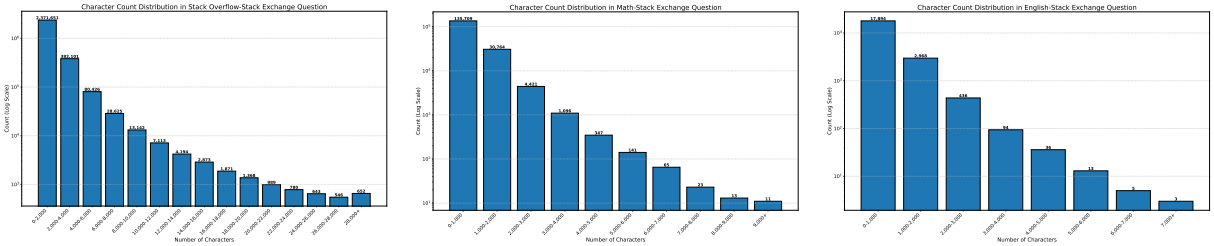


Figure 11: Number of questions by StackExchange category (log scale).



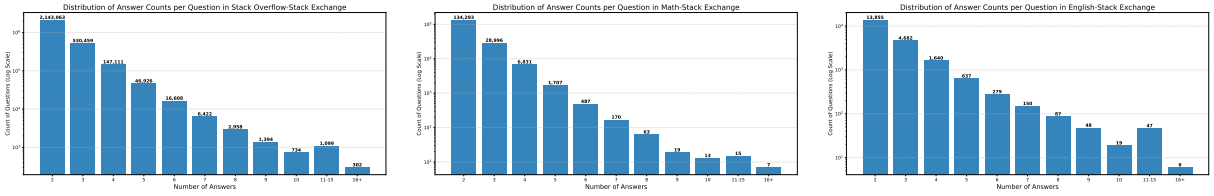
(a) StackOverflow (b) Mathematics (c) English Language & Usage

Figure 12: Number of questions per month in StackOverflow, Mathematics and English Language & Usage.



(a) StackOverflow (b) Mathematics (c) English Language & Usage

Figure 13: Character Count Distribution in StackOverflow, Mathematics and English Language & Usage Questions.



(a) StackOverflow (b) Mathematics (c) English Language & Usage

Figure 14: Distribution of Answer Counts per Question in Mathematics and English Language & Usage.

B.3 TIC-CODEDOCS

Our TIC-CODEDOCS dataset consists of official documentation extracted systematically from the Git repositories of NumPy (Harris et al., 2020) and PyTorch (Ansel et al., 2024).

As shown in Table 4, we include 16 major releases of NumPy (from `v1.13.0` in 2017 to `v2.1.0` in 2024) and 11 major releases of PyTorch (from `v1.8.0` in 2021 to `v2.4.0` in 2024). For each release, we revert the repository to the corresponding commit, build the library in a dedicated virtual environment, and generate HTML documentation via Sphinx. To extract clean plaintext content from these HTML documents, we use `readability.js`[11], which effectively preserves only the main textual content.

The processed documentation is then formatted into a structured JSONL file, where each entry consists of the document's title and textual content. This approach ensures consistency, reproducibility, and accurate representation of each library's documentation, enabling reliable evaluation of model performance (measured by perplexity, $\text{ppl}_{\text{token}}$) across library versions and their temporal evolution.

Table 4: TIC-CODEDOCS: Details of versions, release times, and number of extracted documents.

Numpy Version	Year/Month	#Documents
1.13.0	2017/06	2072
1.14.0	2018/01	2097
1.15.0	2018/07	2111
1.16.0	2019/01	2112
1.17.0	2019/07	2201
1.18.0	2019/12	2450
1.19.0	2020/06	2462
1.20.0	2021/01	2419
1.21.0	2021/06	2502
1.22.0	2021/12	2455
1.23.0	2022/06	2470
1.24.0	2022/12	2538
1.25.0	2023/06	2550
1.26.0	2023/09	2550
2.0.0	2024/06	2510
2.1.0	2024/08	2528

PyTorch Version	Year/Month	#Documents
1.8.0	2021/03	1373
1.9.0	2021/06	2998
1.10.0	2021/10	3127
1.11.0	2022/03	3533
1.12.0	2022/06	3591
1.13.0	2022/10	3665
2.0.0	2023/03	3944
2.1.0	2023/10	4050
2.2.0	2024/01	4096
2.3.0	2024/04	4295
2.4.0	2024/07	4385

B.4 Evaluation metrics

Here, we more comprehensively discuss details about our evaluation metrics. To recap, each run produces a $T_t \times T_e$ matrix of evaluations E where T_t, T_e are the number of training/evaluation timesteps, $E_{i,j}$ is the performance of the model trained after seeing data up to month i and evaluated on the month j.

To reduce the computation costs of running evaluations, while we train and produce checkpoints on all 114 CC dumps in TIC-CC, we evaluate on only 12 checkpoints which are roughly annually spaced. These include the first month from each year between 2014-2024 as well as the very last timestamp July-2024.

To control for inherent difficulty differences across evaluation months, we measure the *regret* $R_{i,j} = E_{i,j} - E^*_j$ where E^*_j is the performance of *Oracle-2024-07* on month j. We subtract E^*_j instead of $E_{j,j}$ since if $E_{j,j}$ is bad it may lead to misleadingly good forward/backward metrics. As stated in Sec. 6, we consider the following summary metrics, first defined assuming $T_t = T_e = T$ (as in the case of TIC-CC where we have perfectly aligned training and evaluation data):

- In-distribution (ID) performance: averages along the matrix diagonal, i.e., $\sum_{i=1}^{T} = R_{i,i}/T$.
- Backward transfer: averages the lower triangular of R, i.e., $\sum_{i=1}^{T} \sum_{j<i} \frac{R_{i,j}}{T(T-1)/2}$.
- Forward transfer: averages the upper triangular of R analogously to backward transfer.

For downstream evaluations where evaluation periods do not align with model checkpoint dates, we use an exclusive assignment strategy to define in-domain (ID) performance. For each checkpoint i, we identify

[11] https://github.com/mozilla/readability

its *nearest preceding evaluation* – the evaluation timestep that comes before or concurrent with checkpoint i and is closest in time. However, since multiple checkpoints may share the same nearest preceding evaluation, we ensure each evaluation timestep is claimed as "concurrent" by only the temporally closest training timestep.

Formally, let t_i and e_j denote the timestamps of checkpoint i and evaluation period j respectively. Checkpoint i contributes $R_{i,j}$ to the ID average only if evaluation period j is both (1) the nearest preceding evaluation to checkpoint i, and (2) checkpoint i is the closest checkpoint to evaluation period j among all checkpoints that have j as their nearest preceding evaluation. All other pairs $R_{i,j}$ contribute to forward transfer when $t_i < e_j$, and backward transfer when $t_i > e_j$.

C Training and hyperparameter details

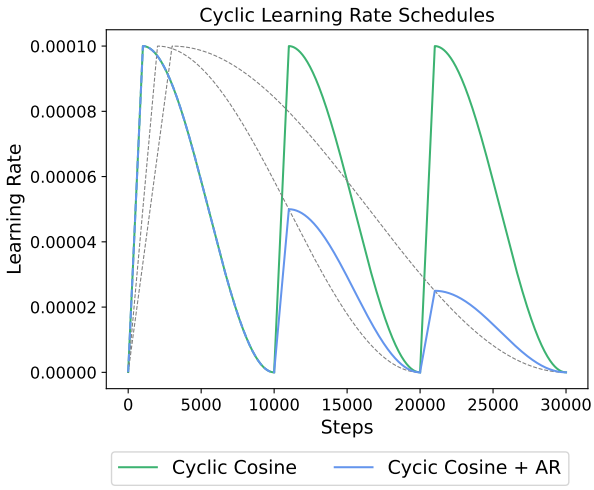


Figure 15: **Cyclic cosine learning rate schedules** compared with a single cosine learning rate schedule (gray) restart the schedule to either the initial maximum (green) or gradually decaying values (blue).

C.1 General details

We follow the architectures and configurations used by DataComp-LM (Li et al., 2024a) for their 3B model scale (unless further specified). For our Oracle and initialization trained on May-2013, we exactly follow their hyperparameters given that these were also standard pre-training runs from scratch. Most notably, this includes using a cosine schedule with maximum learning rate of 3e-3, a final learning rate of 3e-5, and a weight decay of 0.033. Each experiment was run using the OpenLM library (Gururangan et al., 2023) on 64 H100 GPUs with a global batch size of 256 and sequence length of 2048 tokens (using the GPT-NeoX (Black et al., 2022) tokenizer). The continual phases of our training runs require 1.5K (for 220B scale) and 4.5K (for 440B scale) total H100 hours, with the exception of LwF/EWC, for which our implementations incurred about a 30% additional cost.

C.2 Hyperparameters for continual methods

For our various continual methods, we do perform additional hyperparameter tuning using the first 10 TiC-CC training sets and held-out validation sets. Following Cha and Cho (2024), we limit the tuning to an earlier limited set of training rounds given that it would be impossible for a practitioner to be able to tune based upon data they have not seen far in the future. We discuss the tuning and hyperparameter choices for specific methods in more detail below. All results are for the 220B token scale.

Cyclic Cosine. We mainly tuned the maximum learning rate in each cycle, trying values between 1e-3 and 3e-5, as shown in Tab. 5. On our tuning set, the best setting across the board was 1e-4. When carrying out these tuning runs to completion on all 113 timesteps, we do observe an important difference in behavior. While 1e-4 continues to offer the best ID performance and strictly dominates all higher settings, lowering it further can be used to trade-off Backward and ID performance. The smallest fixed max learning rate, 3e-5 results in a similar yet overall worse performance profile to using an an AR meta-schedule. This makes sense given the AR schedule roughly can be considered to decrease the maximum learning rate at a $1/t$ rate; since our setup involves over 100 months, AR schedules set the maximum learning rate very close to the minimum of 3e-5 in most rounds. Overall, we find that learning rates do need to be lowered by at least 30x compared to the the May-2013 initialization (which used 3e-3). This is in contrast to Ibrahim et al. (2024); Gupta et al. (2023) which both suggest re-warming up to a similar learning rate as the initial pre-training or Parmar et al. (2024) who start from the minimum learning rate of the pre-trained model. We suspect this is due to the difference in setup (i.e., these works use only 2 or 3 training rounds of comparable sizes and face distribution shifts related to data quality and language rather than temporal evolution).

Table 5: **Tuning LR for Cyclic Cosine**

Max LR	TIC-CC (Tuning Months)			TIC-CC (All Months)		
	Backward	ID	Forward	Backward	ID	Forward
1e-3	0.103	0.086	0.118	0.197	0.083	0.209
3e-4	0.019	0.016	0.051	0.125	0.041	0.178
1e-4	**0.002**	**0.005**	**0.039**	0.072	**0.027**	**0.161**
5e-5	**0.002**	0.006	**0.039**	0.062	0.034	0.163
3e-5	0.004	0.009	0.040	0.060	0.042	0.165
AR Schedule	**0.002**	0.008	0.043	**0.058**	0.040	0.166

Rsqrt. We tuned both the maximum learning rate within the same range as Cyclic Cosine as well as the cooldown length, choosing between 50 and 400. Our final run continued to use 1e-4 for the maximum learning rate and 400 for the cooldown, though there did not appear to be much difference when compared to smaller values such as 200 or 100 on the tuning months.

Schedule-Free. We continued to use warmup but given that Schedule-Free makes more drastic changes to optimization (i.e., using a different optimizer versus simply a different learning rate schedule), we re-tuned both the learning rate and weight decay. Interestingly, 1e-4 as the maximum learning rate continued to work best for us, though we found it helped slightly to drop the weight decay from 0.033 to 0.01.

Table 6: **Tuning for Schedule-Free**

Max LR	WD	TIC-CC (Tuning Months)		
		Backward	ID	Forward
1e-3	0.033	0.1025	0.0856	0.1178
5e-4	0.033	0.0448	0.0373	0.0713
3e-4	0.033	0.0206	0.0183	0.0532
5e-5	0.033	0.0053	0.0105	0.0406
1e-4	0.067	0.0049	0.0080	0.0406
1e-4	0.033	0.0044	0.0077	0.0404
1e-4	0.010	**0.0042**	**0.0075**	**0.0403**
1e-4	0.005	**0.0042**	**0.0075**	**0.0403**
1e-4	0.0001	0.0044	0.0077	0.0404

LwF. Following the original paper (Li and Hoiem, 2018), we used a temperature parameter of $T = 2$. We mainly tuned the regularization weight λ trying values between 0.1 and 10.0 and settling upon 0.3. However, overall we found using LwF either resulted in little difference (when using a small λ) or started to decrease all metrics (when using a larger λ).

EWC. We fixed the number of iterations used to estimate the Fisher matrix to 100 and similar to LwF, we focused on tuning the weight given to the EWC regularization term. Overall, we found that fairly high values were needed to overcome the small values in the approximate Fisher matrix (coming from small second order moment terms). We found that $\lambda = 10^7$ performed best when tuning between 10^1 and 10^9, as shown in Tab. 7. The only other setting we tried that is not strictly dominated by this choice was $\lambda = 10^6$, which resulted in slightly better ID performance but significantly worse backward transfer.

Table 7: **Tuning λ for EWC**

λ	TIC-CC (Tuning Months)		
	Backward	ID	Forward
10^0	0.0025	0.0050	0.0394
10^1	0.0025	0.0050	0.0394
10^4	0.0025	0.0050	0.0394
10^5	0.0025	0.0049	0.0394
10^6	0.0021	**0.0047**	0.0391
10^7	**0.0013**	0.0050	**0.0389**
10^8	0.0107	0.0178	0.0462
10^9	0.0286	0.0400	0.0586

D Extended Results

D.1 TiC-CommonCrawl (TIC-CC) evaluations

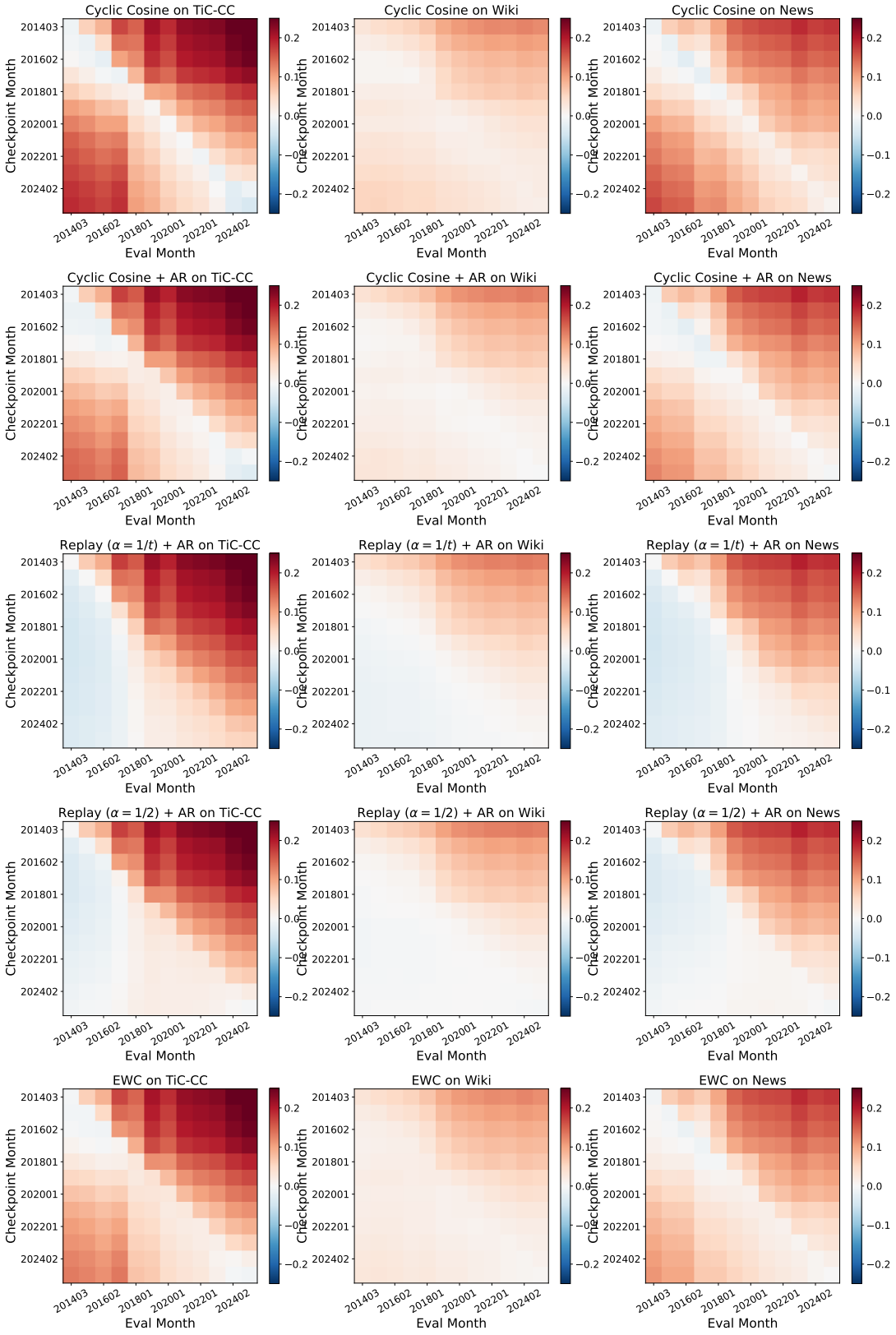


Figure 16: Evaluation matrix heatmaps for selected methods on our TIC-CC evaluations (440B token scale).

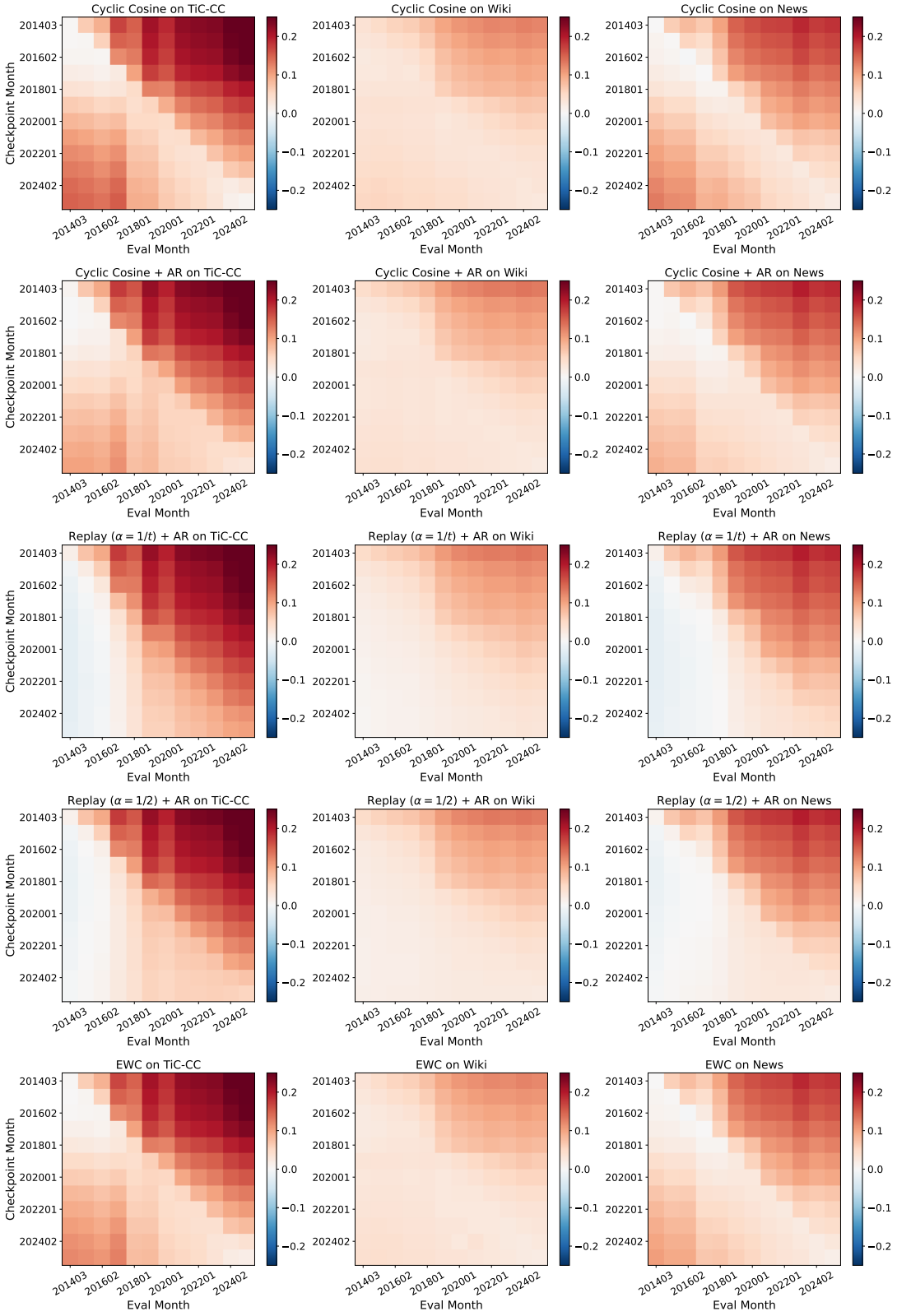


Figure 17: Evaluation matrix heatmaps for selected methods on our TIC-CC evaluations (220B token scale).

D.2 TiC-Wikipedia (TiC-Wiki)

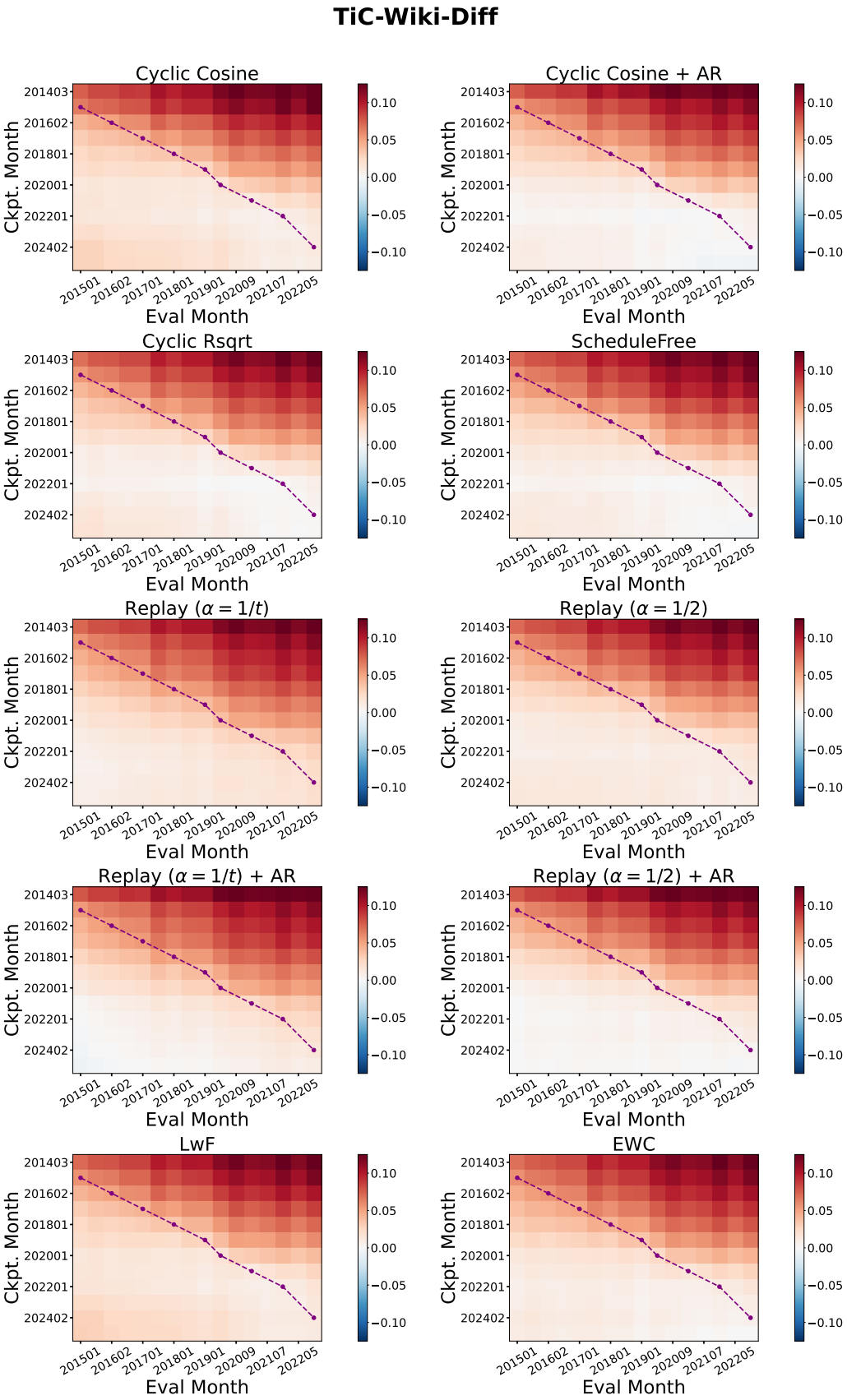


Figure 18: Evaluation matrix heatmaps for various methods on TiC-Wiki-Diff (440B token scale).

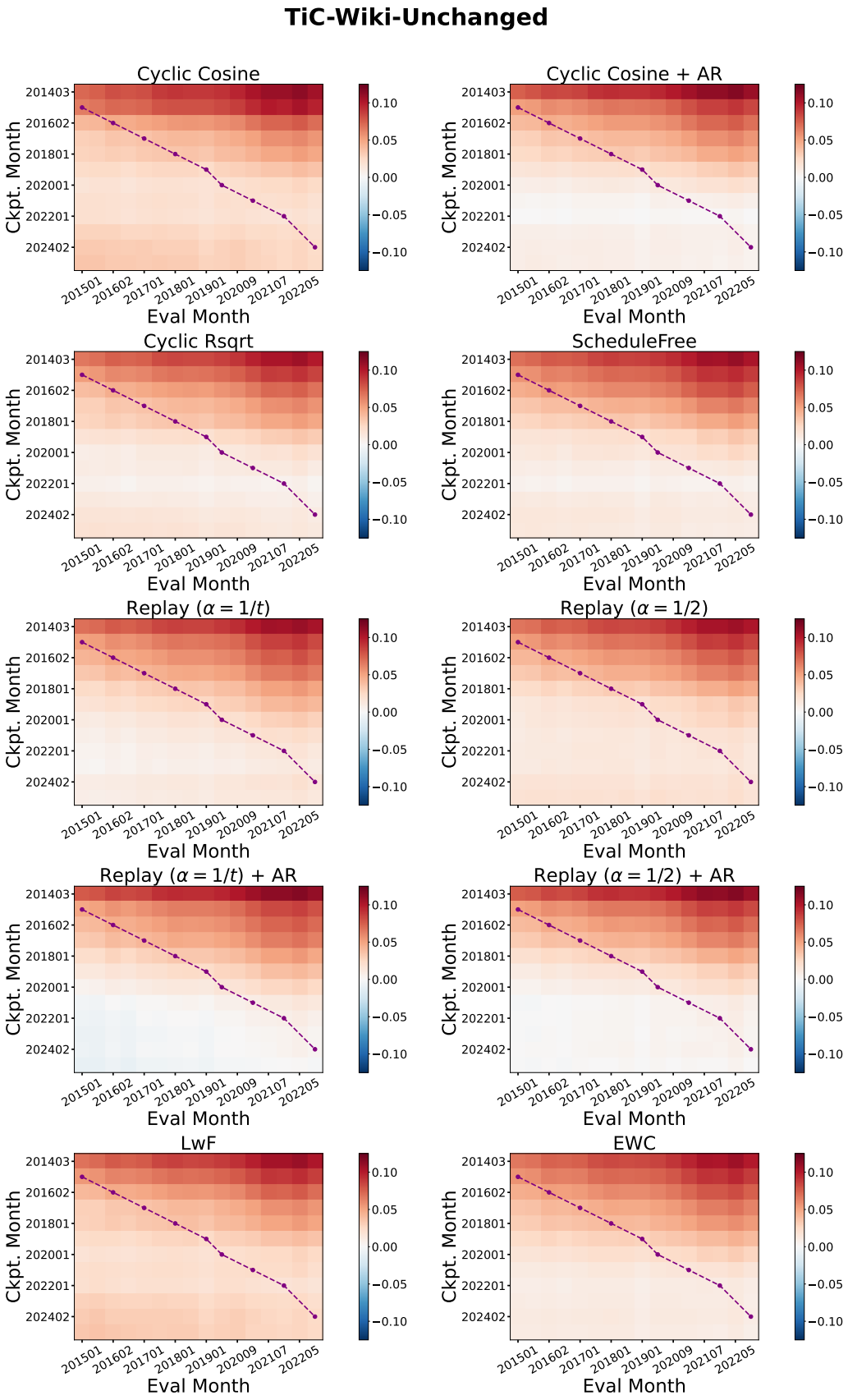


Figure 19: Evaluation matrix heatmaps for various methods on TIC-WIKI-Unchanged (440B token scale).

D.3 TiC-Stackexchange (TIC-STACKE)

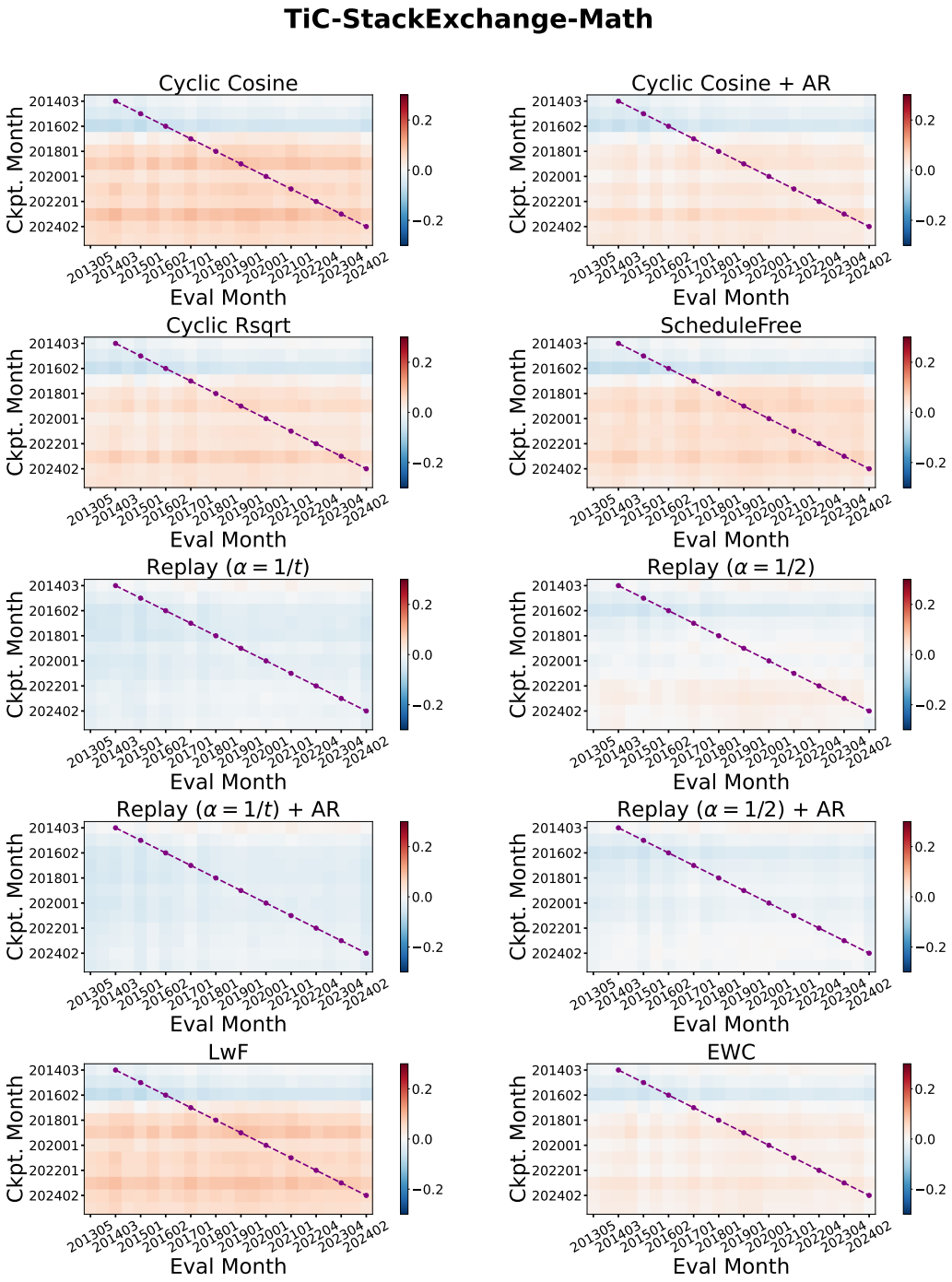


Figure 20: Evaluation matrix heatmaps for various methods on the Math site of TIC-STACKE.

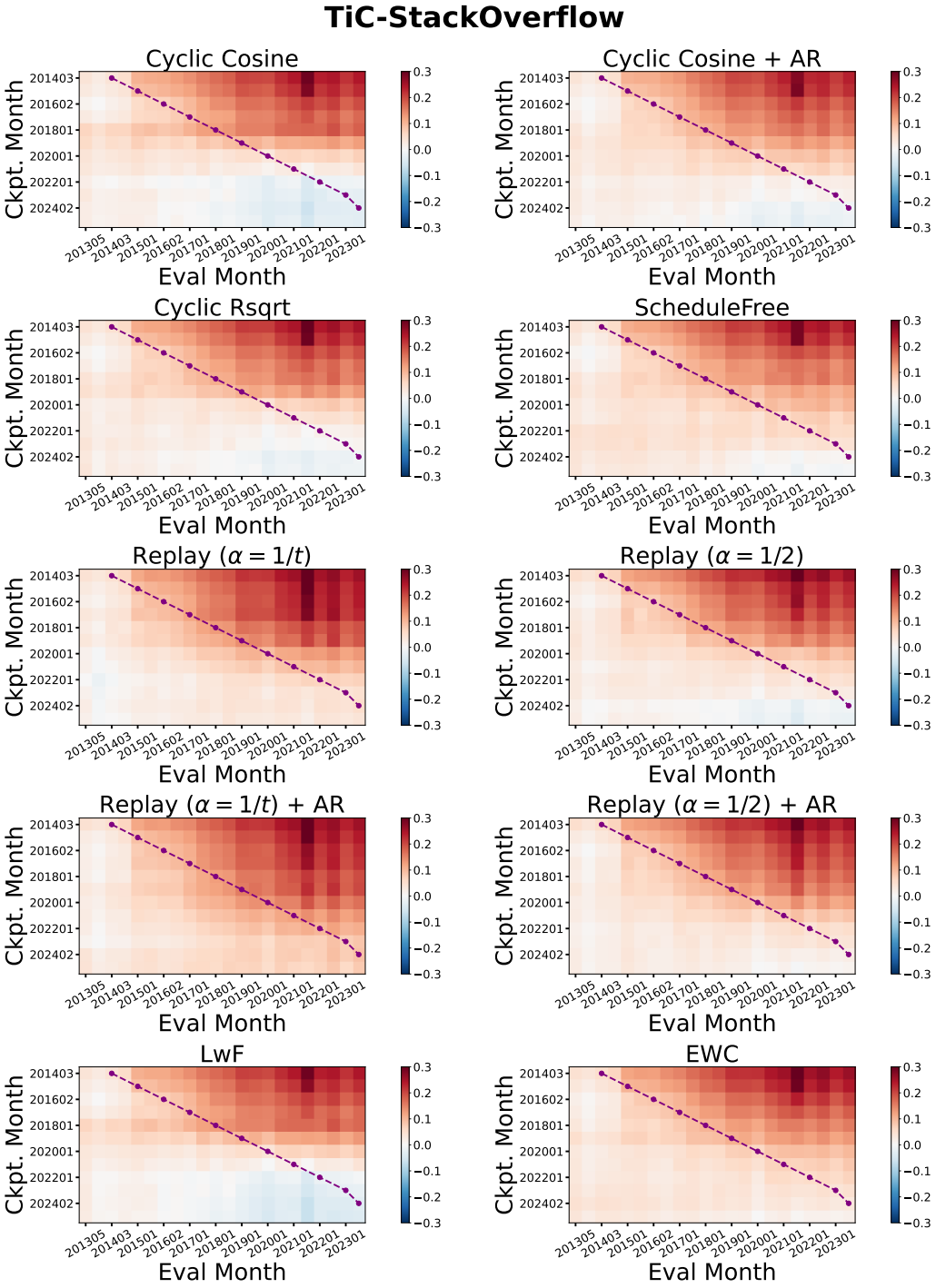


Figure 21: Heatmaps for various methods on the StackOverflow site of TIC-STACKE (440B token scale).

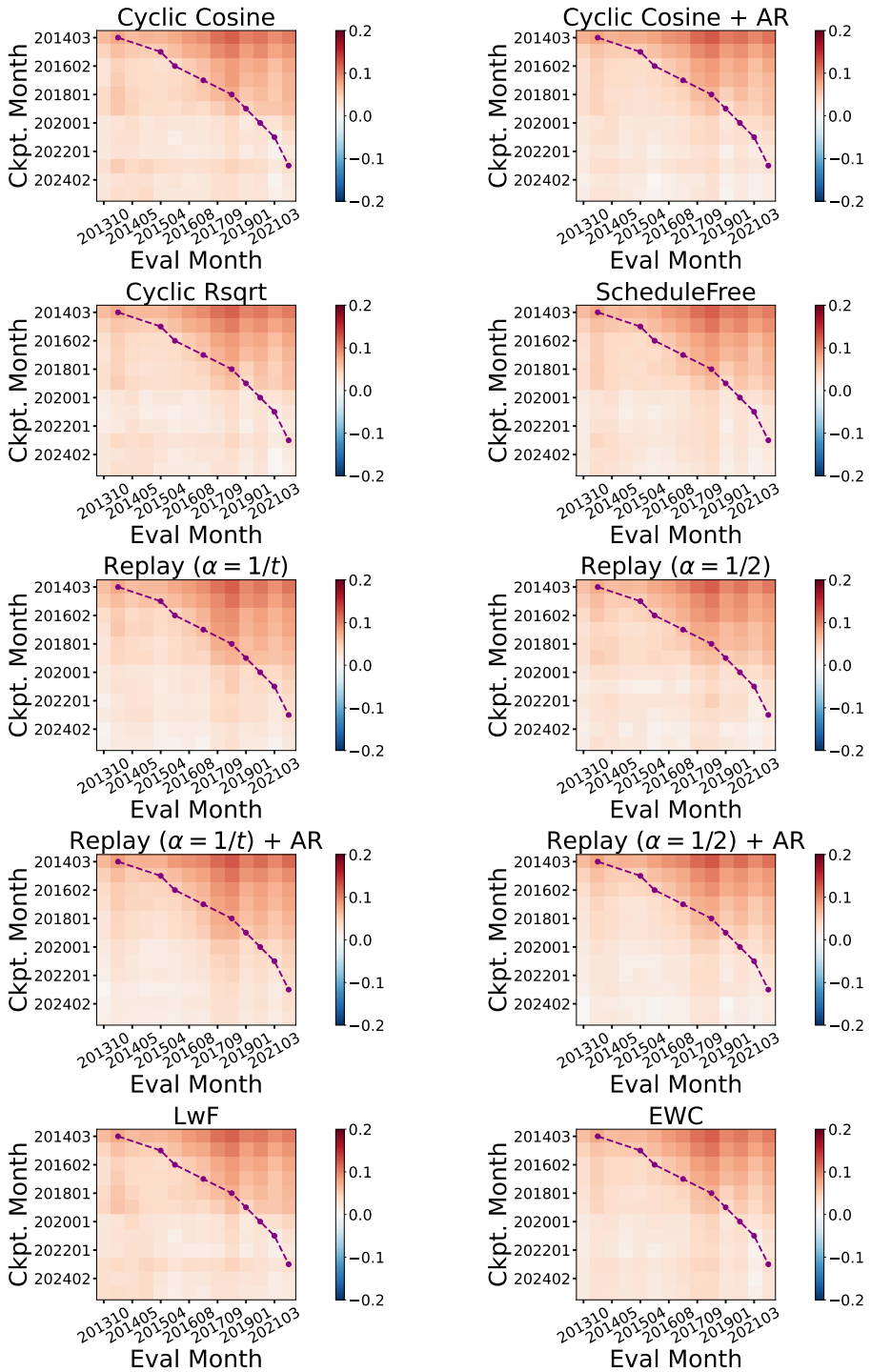


Figure 22: Heatmaps for various methods on the English site of TIC-STACKE (440B token scale).

D.4 TiC-CodeDocs

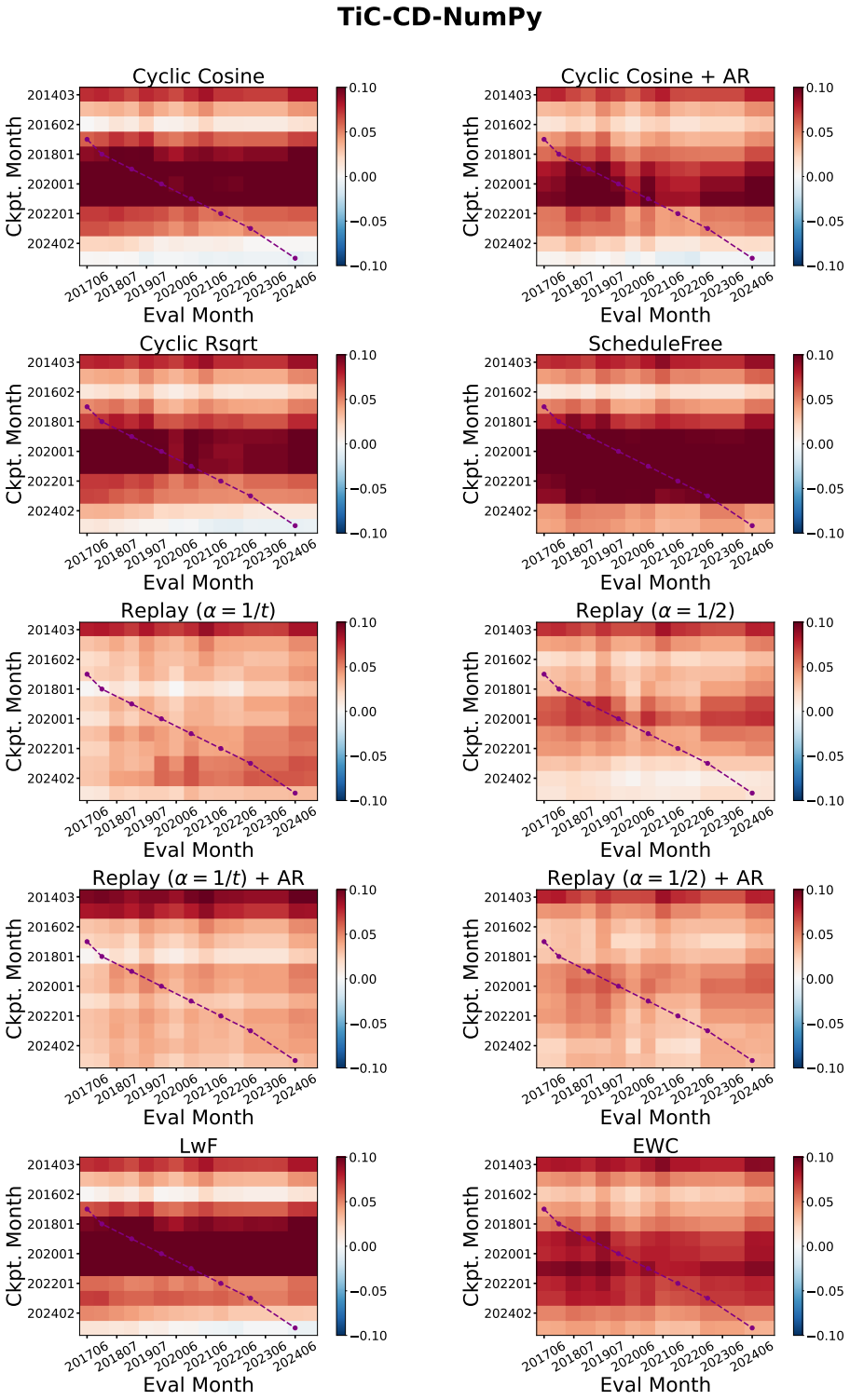


Figure 23: Heatmaps for various methods on TiC-CodeDocs-NumPy (440B token scale).

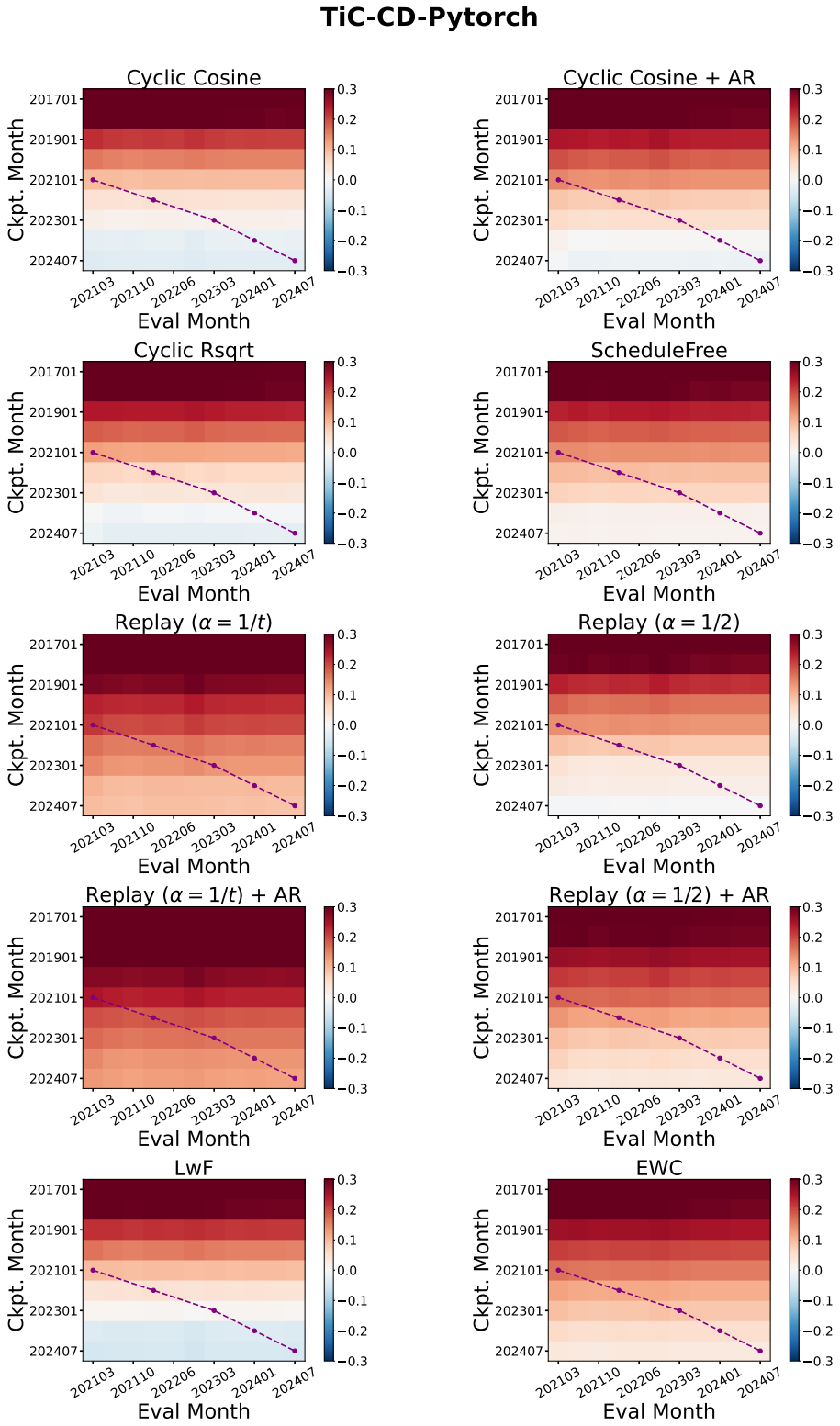


Figure 24: Heatmaps for various methods on TIC-CODEDOCS-PYTORCH (440B token scale).

D.5 TIC-STACKE-English

Table 8: Results for TIC-STACKE-English (3B models, 440B training tokens)

Method	Tokens	TiC-StackE-English ↓ Bwd.	ID	Fwd.
Cyclic Cosine	440B	0.035 (0.002)	0.044 (0.002)	0.070 (0.000)
Cyclic Cosine + AR	440B	0.028	**0.042**	0.070
Cyclic Rsqrt	440B	0.028	**0.041**	**0.068**
ScheduleFree	440B	0.029	0.044	0.071
Replay ($\alpha = 1/t$)	440B	0.030	0.050	0.074
Replay ($\alpha = 1/2$)	440B	0.030	0.045	0.072
Replay ($\alpha = 1/t$) + AR	440B	0.030	0.052	0.075
Replay ($\alpha = 1/2$) + AR	440B	**0.026**	0.045	0.070
LwF	440B	0.032	**0.042**	0.070
EWC	440B	0.030	**0.043**	0.069
Oracle Series	1.16T	0.018	0.037	0.074

D.6 Extended replay exploration

Here, we show results for additional fixed settings for α_t other on top of $1/t$ and 0.5 which we focused on in the main paper. We also try another variant called Replay (Exp), which allocates 50% of the budget to the latest month and then exponentially decreasing percentages to groups of earlier months. Specifically, we chunk all previous months into groups of 10. The most recent 10 months together receive 25% of the overall monthly budget (each contributing 2.5%). Each next oldest group of 10 is allocated half the budget of the previous 10 (with the exception of the last group which makes up the remainder).

Overall, these results largely reinforce the conclusions conclusions discussed in Sec. 6. Different evaluations benefit from more/less replay. On TiC-CC subsets, we see trade-offs between Backward and ID, where more replay helps on Backward but does worse on ID. On downstream evals, the optimal α_t varies. Even more replay helps more on the slower-changing domains (e.g., TiC-CodeDocs-NumPy) and less replay helps on faster moving ones (e.g., TiC-CodeDocs-PyTorch). Using $\alpha_t = 1/t$ is generally sub-optimal, as it is often dominated by $\alpha = 0.1$ on all three metrics. Meanwhile, Replay (Exp) tends to fall somewhere between using $\alpha_t = 0.5$ and $\alpha_t = 0.9$ (as expected) but is most often dominated by simply using $\alpha_t = 0.7$. Given the trade-off across evaluations, among fixed choices of α_t, using a middling value like 0.5 (or slightly higher) seems to be a reasonable practical recommendation to avoid performing poorly on any one evaluation/domain.

Table 9: **Further exploration of replay variants.** Results are for 3B models trained for 220B total tokens.

Replay Variant	TIC-CC			TIC-CC-WIKI			TIC-CC-NEWS		
	Backward	ID	Forward	Backward	ID	Forward	Backward	ID	Forward
Replay ($\alpha = 1/t$)	0.023	0.074	0.178	0.020	0.036	0.078	0.005	0.035	0.117
Replay (Exp)	0.037	0.038	0.165	0.030	0.032	**0.074**	0.027	0.018	0.111
Replay ($\alpha = 0.1$)	**0.022**	0.066	0.175	0.021	0.034	0.077	0.006	0.030	0.116
Replay ($\alpha = 0.3$)	**0.022**	0.052	0.170	0.022	0.032	0.075	0.009	0.022	0.113
Replay ($\alpha = 0.5$)	0.024	0.042	0.167	0.024	**0.031**	**0.074**	0.013	0.019	0.111
Replay ($\alpha = 0.7$)	0.028	0.035	0.164	0.027	**0.031**	**0.074**	0.019	0.017	0.110
Replay ($\alpha = 0.9$)	0.039	**0.029**	**0.162**	0.032	**0.031**	**0.074**	0.030	**0.015**	**0.109**
Replay ($\alpha = 1/t$) + AR	0.026	0.083	0.181	**0.019**	0.037	0.079	**0.004**	0.039	0.119
Replay (Exp) + AR	0.032	0.051	0.170	0.026	0.032	0.076	0.018	0.021	0.112
Replay ($\alpha = 0.1$) + AR	0.026	0.076	0.179	**0.019**	0.036	0.079	0.005	0.035	0.118
Replay ($\alpha = 0.3$) + AR	0.025	0.064	0.174	0.020	0.034	0.077	0.006	0.027	0.114
Replay ($\alpha = 0.5$) + AR	0.025	0.055	0.171	0.022	0.032	0.076	0.009	0.022	0.112
Replay ($\alpha = 0.7$) + AR	0.027	0.048	0.169	0.023	0.032	0.075	0.012	0.020	0.112
Replay ($\alpha = 0.9$) + AR	0.034	0.043	0.167	0.026	**0.031**	0.075	0.019	0.018	0.111

Replay Variant	TIC-WIKI-Diff			TIC-WIKI-Unchanged		
	Backward	ID	Forward	Backward	ID	Forward
Replay ($\alpha = 1/t$)	0.038	0.057	0.091	0.035	0.050	0.074
Replay (Exp)	0.033	0.046	0.086	0.037	0.048	**0.072**
Replay ($\alpha = 0.1$)	0.036	0.054	0.090	0.035	0.050	0.075
Replay ($\alpha = 0.3$)	0.034	0.050	0.088	0.034	0.048	0.074
Replay ($\alpha = 0.5$)	0.034	0.050	0.088	0.034	0.048	0.074
Replay ($\alpha = 0.7$)	**0.032**	0.046	**0.085**	0.036	0.048	**0.072**
Replay ($\alpha = 0.9$)	0.033	**0.044**	**0.085**	0.039	0.048	**0.072**
Replay ($\alpha = 1/t$) + AR	0.039	0.060	0.092	0.034	0.052	0.077
Replay (Exp) + AR	0.033	0.048	0.088	0.034	**0.046**	0.073
Replay ($\alpha = 0.1$) + AR	0.038	0.056	0.091	0.034	0.049	0.076
Replay ($\alpha = 0.3$) + AR	0.036	0.052	0.090	**0.033**	0.047	0.075
Replay ($\alpha = 0.5$) + AR	0.034	0.050	0.088	**0.033**	0.047	0.074
Replay ($\alpha = 0.7$) + AR	0.033	0.048	0.087	**0.033**	**0.046**	0.073
Replay ($\alpha = 0.9$) + AR	0.033	0.046	0.086	0.035	0.047	**0.072**

Replay Variant	TIC-STACKOVERFLOW			TIC-STACKE-Math		
	Backward	ID	Forward	Backward	ID	Forward
Replay ($\alpha = 1/t$)	0.075	0.119	0.191	-0.009	-0.010	0.006
Replay (Exp)	0.054	0.088	0.170	0.026	0.013	0.006
Replay ($\alpha = 0.1$)	0.062	0.104	0.183	-0.007	-0.008	-0.004
Replay ($\alpha = 0.3$)	0.060	0.100	0.176	-0.001	-0.002	-0.001
Replay ($\alpha = 0.5$)	0.055	0.090	0.170	0.010	0.002	0.002
Replay ($\alpha = 0.7$)	0.050	0.086	0.169	0.018	0.009	0.005
Replay ($\alpha = 0.9$)	0.042	**0.076**	**0.160**	0.028	0.016	0.007
Replay ($\alpha = 1/t$) + AR	0.066	0.116	0.193	**-0.019**	**-0.015**	**-0.008**
Replay (Exp) + AR	0.044	0.088	0.174	0.005	0.000	0.000
Replay ($\alpha = 0.1$) + AR	0.057	0.107	0.188	**-0.018**	-0.013	-0.006
Replay ($\alpha = 0.3$) + AR	0.052	0.099	0.179	-0.012	-0.010	-0.004
Replay ($\alpha = 0.5$) + AR	0.047	0.091	0.176	-0.006	-0.006	-0.002
Replay ($\alpha = 0.7$) + AR	0.042	0.086	0.170	-0.001	-0.003	-0.002
Replay ($\alpha = 0.9$) + AR	**0.033**	**0.076**	**0.161**	0.005	0.000	-0.001

Replay Variant	TIC-CODEDOCS-NUMPY			TIC-CODEDOCS-PYTORCH		
	Backward	ID	Forward	Backward	ID	Forward
Replay ($\alpha = 1/t$)	0.054	0.055	0.057	0.172	0.183	0.275
Replay (Exp)	0.070	0.068	0.063	0.069	0.087	0.233
Replay ($\alpha = 0.1$)	0.042	0.045	**0.051**	0.163	0.171	0.268
Replay ($\alpha = 0.3$)	0.056	0.059	0.057	0.130	0.139	0.250
Replay ($\alpha = 0.5$)	0.058	0.062	0.060	0.097	0.110	0.237
Replay ($\alpha = 0.7$)	0.066	0.073	0.068	0.087	0.101	0.232
Replay ($\alpha = 0.9$)	0.069	0.072	0.064	**0.066**	**0.083**	**0.225**
Replay ($\alpha = 1/t$) + AR	0.040	0.042	0.057	0.195	0.202	0.277
Replay (Exp) + AR	0.053	0.052	0.062	0.099	0.114	0.240
Replay ($\alpha = 0.1$) + AR	**0.031**	**0.032**	**0.049**	0.184	0.190	0.271
Replay ($\alpha = 0.3$) + AR	**0.035**	**0.036**	**0.050**	0.155	0.161	0.254
Replay ($\alpha = 0.5$) + AR	**0.035**	**0.038**	0.052	0.127	0.138	0.246
Replay ($\alpha = 0.7$) + AR	0.044	0.046	0.056	0.111	0.121	0.238
Replay ($\alpha = 0.9$) + AR	0.048	0.045	0.054	0.098	0.112	0.234

D.7 Deduplicating newer against older data

In this section, we explore an alternative approach to data deduplication. As mentioned in Section 3, for all other experiments, we performed deduplication only within each month. Here, we consider the effects of also deduplicating newer months against older months. Specifically, we implement this by simply retaining the state of the Bloom Filter across months, only resetting it once every 10 months, which we refer to as a "windowed" deduplication.

Table 10: **Exploration of windowed deduplication (3B models, 440B training tokens).** Here "Loc." refers to the original within-month *local* deduplication strategy and "Wind." refers to our 10-month windowed deduplication.

Method	Dedup	TiC-Wiki-Diff ↓			TiC-Wiki-Unch. ↓			TiC-StackOverflow ↓			TiC-CD-PyTorch↓		
		Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	Bwd.	ID	Fwd.
Replay ($\alpha_t = 1/2$)	Loc.	0.015	0.029	0.073	0.017	0.027	0.055	**0.027**	0.065	0.158	**0.020**	**0.032**	**0.189**
Replay ($\alpha_t = 1/2$) + AR	Loc.	0.010	0.027	0.073	0.007	0.022	0.054	0.036	0.074	0.160	0.058	0.070	0.214
Replay ($\alpha_t = 1/2$)	Wind.	0.013	0.026	0.072	0.014	0.023	0.052	0.029	**0.059**	**0.149**	0.036	0.048	0.199
Replay ($\alpha_t = 1/2$) + AR	Wind.	**0.007**	**0.024**	**0.070**	**0.002**	**0.017**	**0.049**	0.037	0.071	0.154	0.064	0.078	0.221
Oracle Series	Loc.	0.014	0.035	0.080	0.013	0.030	0.061	0.012	0.056	0.146	0.035	0.057	0.196

Method	Dedup	TiC-StackE-Math ↓			TiC-CD-NumPy ↓			Static Evals. ↑
		Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	CORE (DCLM)
Replay ($\alpha_t = 1/2$)	Loc.	0.000	-0.008	-0.011	**0.029**	**0.038**	0.044	50.1
Replay ($\alpha_t = 1/2$) + AR	Loc.	**-0.015**	**-0.019**	**-0.017**	0.036	0.039	**0.043**	49.2
Replay ($\alpha_t = 1/2$)	Wind.	0.011	0.002	-0.001	0.085	0.088	0.086	49.8
Replay $\alpha_t = 1/2$) + AR	Wind.	-0.002	-0.009	-0.008	0.078	0.078	0.078	49.5
Oracle Series	Loc.	-0.025	-0.028	-0.022	0.008	0.008	0.015	50.6

In Table 10, we apply the new deduplication strategies and measure performance on downstream evaluations only (as TIC-CC evaluations will be biased towards the original within-month deduplication as it was used to construct TIC-CC test sets). Overall, we observe that deduplicating across months, when applied with replay, can help improve performance on some evaluations but not all. We speculate that this is because on rarer domains, such as code, deduplicating more aggressively across months can reduce the already limited amount of relevant samples. On the other hand, for evaluations where more general data helps, such as TIC-WIKI, deduplication helps by letting the model see more diverse data given the same token budget.

D.8 Effects of model size on downstream evaluations.

Table 11: **Comparing downstream evaluations for 1B and 3B models (220B training tokens).** For all dynamic evaluations, we report perplexity values relative to the *Oracle-2024-07* model of the same size and with log-scaling. Meanwhile, CORE is an average of the accuracies of 22 downstream zero/few-shot tasks used by Li et al. (2024a), evaluated only on the final model checkpoint (without scaling by an Oracle). **Bold** values are within one standard deviation (estimated with 3 runs of Cyclic Cosine) of the best in each column for a given model size.

Method	Params	TiC-Wiki-Diff ↓			TiC-Wiki-Unch. ↓			TiC-StackOverflow ↓			TiC-CD-PyTorch↓		
		Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	Bwd.	ID	Fwd.
Cyclic Cosine (std)	3B	0.033 (0.000)	0.045 (0.000)	0.085 (0.000)	0.039 (0.000)	0.048 (0.000)	0.072 (0.000)	0.041 (0.002)	0.073 (0.002)	**0.156** (0.003)	**0.057** (0.002)	**0.072** (0.002)	0.219 (0.002)
Cyclic Cosine + AR	3B	0.033	0.048	0.087	0.035	0.047	0.074	**0.032**	**0.072**	0.159	0.081	0.096	0.228
Cyclic Rsqrt	3B	**0.031**	**0.043**	0.084	0.035	**0.045**	0.070	0.035	**0.071**	**0.158**	0.059	0.074	0.220
Schedule-Free	3B	0.035	0.048	0.087	0.040	0.050	0.074	0.038	0.074	0.160	0.081	0.096	0.236
Replay ($\alpha_t = 1/t$)	3B	0.038	0.057	0.091	0.035	0.050	0.074	0.075	0.119	0.191	0.172	0.183	0.275
Replay ($\alpha_t = 1/2$)	3B	0.032	0.047	0.086	0.034	0.047	0.072	0.055	0.090	0.170	0.097	0.110	0.237
Replay ($\alpha_t = 1/t$) + AR	3B	0.039	0.060	0.092	0.034	0.052	0.077	0.066	0.116	0.193	0.195	0.202	0.277
Replay ($\alpha_t = 1/2$) + AR	3B	0.034	0.050	0.088	**0.033**	0.047	0.074	0.047	0.091	0.176	0.127	0.138	0.246
LwF	3B	0.033	0.045	0.085	0.039	0.048	0.072	0.037	**0.070**	**0.155**	**0.055**	**0.070**	**0.216**
EWC	3B	**0.031**	0.044	**0.083**	0.034	**0.045**	**0.069**	**0.033**	**0.072**	0.162	0.067	0.080	0.222
Oracle Series	3B	0.014	0.035	0.080	0.013	0.030	0.061	0.012	0.056	0.146	0.035	0.057	0.196

Method	Params	TiC-StackE-Math ↓			TiC-CD-NumPy ↓			Static Evals. ↑
		Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	CORE (DCLM)
Cyclic Cosine (std)	3B	0.037 (0.001)	0.024 (0.001)	0.015 (0.000)	0.070 (0.004)	0.075 (0.006)	0.070 (0.002)	48.5 (0.4)
Cyclic Cosine + AR	3B	0.011	0.006	0.003	0.055	0.056	0.062	48.5
Cyclic Rsqrt	3B	0.024	0.016	0.009	0.067	0.072	0.071	**49.0**
Schedule-Free	3B	0.035	0.025	0.017	0.070	0.075	0.080	**48.8**
Replay ($\alpha_t = 1/t$)	3B	-0.009	-0.010	-0.006	0.054	0.055	0.057	**48.9**
Replay ($\alpha_t = 1/2$)	3B	0.010	0.002	0.002	0.058	0.062	0.060	**49.0**
Replay ($\alpha_t = 1/t$) + AR	3B	**-0.019**	**-0.015**	**-0.008**	0.040	**0.042**	0.057	**49.0**
Replay ($\alpha_t = 1/2$) + AR	3B	-0.006	-0.006	-0.002	**0.035**	**0.038**	**0.052**	**49.2**
LwF	3B	0.034	0.022	0.013	0.073	0.078	0.073	48.5
EWC	3B	0.016	0.010	0.006	0.056	0.060	0.067	**48.9**
Oracle Series	3B	-0.025	-0.028	-0.022	0.008	0.008	0.015	50.6

Method	Params	TiC-Wiki-Diff ↓			TiC-Wiki-Unch. ↓			TiC-StackOverflow ↓			TiC-CD-PyTorch↓		
		Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	Bwd.	ID	Fwd.
Cyclic Cosine (std)	1B	0.035 (0.000)	0.044 (0.000)	0.079 (0.000)	0.040 (0.000)	0.045 (0.000)	0.064 (0.000)	**0.074** (0.001)	**0.113** (0.001)	**0.211** (0.001)	**0.061** (0.004)	**0.079** (0.004)	**0.232** (0.004)
Cyclic Cosine + AR	1B	0.035	0.045	0.081	0.035	0.043	0.065	0.089	0.135	0.222	0.104	0.120	0.252
Cyclic Rsqrt	1B	0.034	0.044	0.079	0.037	0.044	0.063	0.084	0.126	0.219	0.082	0.099	0.245
ScheduleFree	1B	0.040	0.049	0.083	0.044	0.050	0.068	0.095	0.138	0.232	0.111	0.127	0.262
Replay ($\alpha = 1/t$)	1B	0.038	0.054	0.085	0.034	0.046	0.066	0.100	0.152	0.236	0.185	0.196	0.284
Replay ($\alpha = 1/2$)	1B	0.036	0.048	0.081	0.036	0.045	0.064	0.089	0.129	0.218	0.115	0.129	0.248
Replay ($\alpha = 1/t$) + AR	1B	0.039	0.055	0.085	**0.032**	0.044	0.066	0.113	0.167	0.245	0.214	0.225	0.299
Replay ($\alpha = 1/2$) + AR	1B	0.036	0.049	0.083	**0.032**	0.043	0.064	0.097	0.145	0.231	0.152	0.165	0.272
LwF	1B	0.036	0.045	0.080	0.041	0.046	0.065	0.076	0.119	0.213	**0.065**	**0.083**	0.237
EWC	1B	**0.033**	**0.043**	**0.078**	0.035	**0.042**	**0.061**	0.086	0.128	0.220	0.097	0.114	0.249
Oracle Series	1B	0.015	0.043	0.073	0.016	0.040	0.058	0.032	0.089	0.176	0.040	0.000	0.201

Method	Params	TiC-StackE-Math ↓			TiC-CD-NumPy ↓			Static Evals. ↑
		Bwd.	ID	Fwd.	Bwd.	ID	Fwd.	CORE (DCLM)
Cyclic Cosine	1B	0.063 (0.001)	0.051 (0.001)	0.034 (0.001)	0.097 (0.003)	0.115 (0.004)	0.123 (0.006)	45.3 (0.003)
Cyclic Cosine + AR	1B	0.042	0.033	0.023	0.093	0.112	0.126	**45.6**
Cyclic Rsqrt	1B	0.054	0.043	0.029	0.096	0.115	0.127	**45.7**
ScheduleFree	1B	0.067	0.055	0.041	0.112	0.132	0.142	**45.6**
Replay ($\alpha = 1/t$)	1B	0.013	0.012	0.011	**0.082**	**0.095**	**0.110**	**45.6**
Replay ($\alpha = 1/2$)	1B	0.032	0.024	0.018	**0.084**	**0.097**	**0.110**	**45.6**
Replay ($\alpha = 1/t$) + AR	1B	**0.008**	**0.010**	**0.011**	0.090	0.103	**0.116**	**45.6**
Replay ($\alpha = 1/2$) + AR	1B	0.021	0.016	0.015	**0.082**	**0.099**	**0.114**	**45.9**
LwF	1B	0.062	0.049	0.034	0.096	0.119	0.130	**45.6**
EWC	1B	0.041	0.033	0.023	0.107	0.123	0.128	**45.6**
Oracle Series	1B	-0.012	-0.017	-0.016	0.016	0.025	0.037	46.7

As seen in Table 11 above, for the same token budget, changing model size can lead to continual methods being more suboptimal compared to *Oracle-2024-07* on all evaluations besides TIC-WIKI. However, the overall conclusions do not majorly change (e.g., which evaluations benefit from replay versus not).

E Extended Related Work

Temporal knowledge evaluations. Various benchmarks have been proposed to evaluate the temporal knowledge of LLMs. TemporalWiki (Jang et al., 2022a) evaluates the capability of models to update factual knowledge. TemporalWiki is constructed from the difference between four consecutive snapshots of Wikipedia and Wikidata. Our TIC-WIKI evaluation expands and improves on TemporalWiki in various ways (see Appx. B.1). StreamingQA (Liška et al., 2022) consists of human written and generated questions from 14 years of news articles. The evaluation is either open-book where a model receives a collection of news articles that contain the answer, or closed-book where the model is first fine-tuned on the training set containing the documents and then tested. TempEL (Zaporojets et al., 2022) evaluates entity linking performance across 10 yearly snapshots of Wikipedia. Entity linking is the task of mapping anchor mentions to target entities that describe them in a knowledge base. In comparison, our TIC-WIKI evaluates general language and knowledge understanding. TempLAMA (Dhingra et al., 2022) constructs an evaluation for factual queries from Wikidata. They focus on temporally sensitive knowledge with known start and end dates in a specific Wikidata snapshot. Notably, they propose TempoT5 to jointly model text and timestamp which allows a language model to answer temporal questions that change over time such "Who is the president". EvolvingQA (Kim et al., 2024) is also a benchmark for training and evaluating on Wikipedia over time where a LLM automatically generates question-answers from 6 months of articles in 2023. We avoid using any LLMs for generating our evaluations to prevent any transfer of biases. TIQ (Jia et al., 2024) benchmark consists of 10k questions-answers based on significant events for the years 1801–2025.

Temporal generalization. Beyond understanding the past, LLMs need to be prepared for the future. Li et al. (2024b) observes performance deterioration of public LLMs on Wikipedia, news, code documentation, and arXiv papers after their training data cutoff date. They particularly use compression rate achieved by treating an LLM as a general input compressor using arithmetic coding (Delétang et al., 2024). Our comprehensive evaluations on CommonCrawl, Wikipedia, news articles, StackExchange, and code documentation evaluations verify their results and more comprehensively show that the rate of deterioration is domain-specific. DyKnow (Mousavi et al., 2024b) evaluations also reaffirm the finding that private and open-source LLMs have outdated knowledge by asking them questions constructed using Wikidata. They also observe LLMs output inconsistent answers in response to prompt variations and current knowledge editing methods do not reduce outdatedness. TAQA (Zhao et al., 2024) further demonstrated that pretrained LLMs mostly answer questions using knowledge from years before their pretraining cutoff. They construct question/answers from Wikipedia for years 2000–2023 and propose three methods to improve the temporal alignment of models. Similar observations have been made in RealTimeQA (Kasai et al., 2024) and TempUN (Beniwal et al., 2024). These works further solidify the need for continuously updating models with continual pretraining.

Temporal understanding. General temporal understanding involves reasoning based on the relation between existing knowledge. Test of Time benchmark (Fatemi et al., 2024) evaluates temporal reasoning, logic, and arithmetic by constructing a synthetic dataset. Their goal is to reduce the chance of factual inconsistency in the evaluation using synthetic data. Our benchmark is designed to be fully realistic based on real data and timestamps to understand the challenges of large-scale continual pretraining in practice. Gurnee and Tegmark (2024) find that LLMs learn a representation of space and time with individual neurons that encode spatial and temporal coordinates. They construct datasets of named entities and find that linear probing LLMs performs well on predicting spatial and temporal coordinates. Nylund et al. (2024) proposed time vectors that specify a direction in the model's weight space that improve performance on text from a specific time period.

Temporal domain-specific evaluations. We can further analyze the temporal understanding of a model based on the performance on specific domains with varying rates of change. Luu et al. (2022) studied temporal misalignment such as quantifying temporal degradation of domain-specific finetuning in four domains: social media, science, news, and food reviews. They observed significant temporal degradation in domains such as news, social media, and science but less in food reviews. Gururangan et al. (2020) studied domain-adaptive pretraining and task-adaptive pretraining on unlabeled data for four domains in

science, news, and reviews. They observe domain/task-adaptive pretraining improves performance on the new domain but do not evaluate forgetting on previous domains. Agarwal and Nenkova (2022) studies the temporal model deterioration on future evaluations. They find that the deterioration is task-dependent and domain-adaptive pretraining does not help hypothesizing that limited pretraining data is detrimental in continual pretraining. Jin et al. (2022) domain-incremental pretraining for four scientific domains as well as temporal pretraining on social media over 6 years. They focus on the impact on downstream performance after fine-tuning. They observe distillation-based approaches are the most effective in retaining dowstream performance for tasks related to earlier domains. Overall, the gap between different continual learning methods remained small that can be due to the small scale of pretraining. In comparison, our TIC-CC training is simulating large-scale pretraining.

Domain/task-continual learning for LLMs. In domain/task continual learning, the model is presented with a sequence of tasks with predefined labels (Hsu et al., 2018; Van de Ven and Tolias, 2019; Zhou et al., 2023). Each task comes with its training and test sets. In contrast with continual pretraining, the model needs to support a growing set of labels while compared with temporal continual learning, the order of tasks are often arbitrary (e.g., Split-CIFAR, Perm-MNIST). Prominent methods in this domain are regularization-based methods (Kirkpatrick et al., 2017; Mirzadeh et al., 2020a,b; Farajtabar et al., 2020), replay-based methods that often perform superior (Lomonaco et al., 2022; Balaji et al., 2020; Prabhu et al., 2020), and architecture-based methods that adapt the model over time (Schwarz et al., 2018; Rusu et al., 2016). Continual learning for language models has also been dominated by domain/task continual works. Jin et al. (2022) proposed benchmarks for continually training models on a sequence of research paper domains as well as chronologically-ordered tweet streams. Razdaibiedina et al. (2023) proposed learning a new soft prompt for each task and pass soft prompts for all seen tasks to the model which provides adaptability while preventing catastrophic forgetting. Luo et al. (2023) studied continual learning for instruction tuning and observed catastrophic forgetting, especially for larger models. Mehta et al. (2023) showed that generic pretraining implicitly reduces catastrophic forgetting during task incremental finetuning.

Continual pretraining of LLMs. Recent work have studied continual pretraining of foundation models at large-scale. TiC-CLIP (Garg et al., 2024) proposed a benchmark of training and evaluation of image-text foundation models and demonstrated the deterioration of existing foundation models on new data. Lazaridou et al. (2021) studied time-stratified language pretraining on WMT, news, and arXiv up to 2019 and observed the models become outdated quickly on news data that holds even for models of various sizes. They study dynamic evaluation as a continual pretraining method that trains on a stream of chronologically ordered documents and observed that models can be updated. However, they did not explore the impact on forgetting and scalability of the method to more generic pretraining data over years. Jang et al. (2022b) proposed continual knowledge learning as a new problem and suggested that parameter expansion is necessary to retain and learn knowledge. They focus on one-step continual pretraining where models are pretrained on C4/Wikipedia data up to 2020 and then trained once more on recent news articles. They find adapter methods perform better than regularization and replay methods. Adapter methods are not directly applicable in our multi-year continual pretraining setup where we train in more than 100 steps on large-scale data. Gupta et al. (2023) proposed simple recipes for continual pretraining of LLMs such as utilizing cyclical learning rate schedules with warmup and ablated on hyperparameters such as warmup duration when continuing the pretraining on a fixed pair of pretraining datasets.

Time-aware training. Orthogonal to continual pretraining, one can modify the training or fine-tuning of a model to include explicit information about time. TempLAMA (Dhingra et al., 2022) proposed prepending a time prefix to each example during training which gives the model the flexibility to respond to time-sensitive questions. They train models on news articles where the time can be reliably extracted. Drinkall et al. (2024) proposed training a series of models with sequential data cutoffs dates to avoid data contamination with benchmark and private data. The observe no difference across time on static downstream evaluations when training models on news and Wikipedia

Factual editing and retrieval augmented generation (RAG). Another set of works aim to address the staleness of pretrained LLMs without further standard pretraining. One approach is to surgically edit the facts a model "knows" by identifying and updating the relevant weights within a model (Mitchell et al.,

2022a). Another is to store edits in an explicit memory and learn to reason over them (Mitchell et al., 2022b). Retrieval augmented generation (RAG) pairs an LLM with new data sources to retrieve the most relevant document for a query. Generally, continual pretraining and RAG are orthogonal approaches to generate up to date responses. RAG methods increase the cost at inference time without changing the model while continual pretraining is the opposite. FreshLLMs (Vu et al., 2024) proposes a QA benchmark and argues that fast-changing knowledge requires a retrieval-based solution compared with slow-changing knowledge. Continual pretraining can be crucial in reducing the cost of RAG by utilizing retrieval only on knowledge that changes faster than the rate of continual pretraining.