Ted-Tok: Maintaining an Evolving Vocabulary for Lifelong Learning

Anonymous ACL submission

Abstract

Lifelong learning investigates how models adapt when exposed to a potentially infinite stream of data. Most conventional approaches focus on updating model parameters (i.e., the neural network weights) as the underlying data distribution evolves over time. However, in natural language processing, model parameters are not the only components that matter. The tokenizer, a foundational part of the system, is usually assumed to remain fixed in lifelong learning scenarios. In this work, we challenge the validity of this assumption: as language evolves, a static tokenizer fragments newly emerging lexical items, reducing compression efficiency and consequently degrading the model performance. We introduce the Temporal Drift Tokenizer (Ted-Tok), which maintains an evolving vocabulary that adapts to emerging linguistic patterns over time. This adaptivity is driven by time-weighted frequency estimators that smooth short-term fluctuations to capture persistent linguistic trends, and a principled addition-deletion strategy targeting sink tokens. Across multiple domains, Ted-Tok consistently improves compression and task performance, with gains increasing under stronger drift, underscoring the role of tokenizer adaptivity in lifelong learning.

1 Introduction

Language is not static: new concepts emerge, usage patterns shift, and meanings evolve over time. Consequently, any NLP models trained on data available at one time will become outdated in the future (Sun et al., 2020; Shi et al., 2024). Handling this temporal drift in the data distribution is one of the core challenges in lifelong learning.

To enable temporal adaptation, most prior work operates at the level of model weights – either by updating parameters during continual training or by injecting temporal signals to encode time-aware representations (Jin et al., 2022; Jang et al., 2022; Su et al., 2023; Dhingra et al., 2022; Rosin et al., 2022). However, a language processing system is more than just its neural network. The tokenizer – responsible for bridging raw text strings and contextual embeddings – plays an equally critical, yet often undervalued role. A suboptimal tokenizer can produce unnatural or semantically meaningless segmentations, thereby increasing training difficulty and degrading model performance. In lifelong learning, as the underlying text distribution evolves, a static tokenizer is increasingly prone to such errors. Consequently, model-weight–level updates alone, as adopted in prior work, are insufficient for achieving temporal adaptation.

To better understand this problem, we first expose the limitations of using a static tokenizer in lifelong learning. To show how static tokenizers become outdated, we build byte-pair encoding (BPE; Sennrich et al. 2016) tokenizers on different time slices of the corpora and observe growing differences in their vocabulary as the time gap increases. We then evaluate the compression efficiency of these tokenizers and find that older tokenizers consistently encode fresh text less efficiently. To further assess whether this shift affects task performance, we train identical models on the same corpus with different tokenizers. Across both synthetic and real-world data, misaligned tokenizers result in worse task performance.

The above observations indicate that lifelong learning cannot rely on a fixed tokenizer. However, standard BPE tokenizers are constructed iteratively from a static corpus and do not provide mechanisms for adding or deleting tokens when data arrives in real-time. To address this, we propose the Temporal Drift Tokenizer (**Ted-Tok**). To capture distribution shifts in continuous data streams, we employ a time-weighted token frequency estimator that smooths short-term fluctuations while preserving stable long-term statistics. Furthermore, by reformulating the BPE merge list as a directed acyclic

graph, we introduce a principled mechanism for adding novel tokens and removing outdated ones via sink-node identification. To avoid rescanning historical data during token deletion, we adopt a buffer-based strategy for frequency recalibration. Together, these techniques enable Ted-Tok to revise its vocabulary both efficiently and effectively during model training.

To assess the effectiveness of Ted-Tok, we conducted experiments on synthetic and real-world tasks. Our results demonstrate that by mitigating misalignment with evolving data distributions, Ted-Tok achieves higher compression efficiency and significantly improves task performance in lifelong learning scenarios.

2 Preliminary

In this work, we primarily focus on developing new techniques for the widely used BPE tokenizer. BPE builds its vocabulary through an iterative process that repeatedly merges the most frequently co-occurring pairs of tokens and updates the vocabulary. For completeness, we briefly introduce the algorithm in this section.

Assume we have a text corpus $\mathcal{D}$, from which we build a vocabulary $\mathcal{V}$ and an ordered merge list $\mathcal{M}$. We initialize $\mathcal{V}$ with a base vocabulary $\mathcal{V}_{base}$, consisting solely of individual characters and special tokens. When constructing a BPE tokenizer, each word in the text corpus is first converted into a sequence of characters. BPE counts all adjacent token pairs $(u, v) \in \mathcal{V} \times \mathcal{V}$ within each word in $\mathcal{D}$ and selects the most frequent one $(u_{\text{new}}, v_{\text{new}})$. Then, a merged token $w_{\text{new}} = \text{CONCAT}(u_{\text{new}}, v_{\text{new}})$ is appended to $\mathcal{V}$, and the corresponding merge rule $(u_{\text{new}}, v_{\text{new}})$ is appended to $\mathcal{M}$. After that, every adjacent token pair $(u_{\text{new}}, v_{\text{new}})$ in the corpus $\mathcal{D}$ is replaced with the merged token w_{new}. This procedure repeats until the vocabulary reaches a target size. When a BPE tokenizer processes a word, it first breaks the word into characters and then scans through the ordered merge list $\mathcal{M}$, applying each merge rule to the sequence, until no further merges can be applied.

3 Temporal Drift of Static Tokenizers

In this section, we demonstrate that as the corpus evolves, the mismatch between the data and a fixed tokenizer grows, which consequently reduces tokenization efficiency and leads to degraded task performance. Detailed experimental settings for this section are provided in Appendix C.

3.1 Empirical Evidence of Vocabulary Drift

To examine temporal drift, we first analyze how the vocabulary evolves over time and how this evolution impacts tokenization efficiency.

Experimental setup. We conduct our analysis on three domains: (i) *English WMT News Crawl*, containing news articles published from 2007 to 2021; (ii) *Amazon Reviews*, consisting of product reviews from 1996 to 2018; and (iii) *ArXiv Abstracts*, comprising scientific abstracts from 2007 to 2025. For each domain, we partition the corpus into temporal slices based on timestamps. Each slice is used to construct its own tokenizer using the same vocabulary size.

To quantify vocabulary drift, we measure the difference between two vocabularies $\mathcal{V}_a$ and $\mathcal{V}_b$ using the Jaccard distance, a standard tool for comparing vocabularies (Sawada and Goyal, 2025; Chelombitko et al., 2024), defined as

$$d_J(\mathcal{V}_a, \mathcal{V}_b) = 1 - \frac{|\mathcal{V}_a \cap \mathcal{V}_b|}{|\mathcal{V}_a \cup \mathcal{V}_b|}$$

To assess tokenization efficiency, we measure bytes per token, defined as the byte length of the input string divided by the number of tokens produced by the tokenizer. Higher values indicate more compact compression.

Findings. Figure 1 shows the Jaccard distance grows steadily as the temporal gap between vocabularies increases. In English WMT News Crawl, the 2007–2009 and 2010–2012 vocabularies remain highly similar (0.186), whereas the distance grows to 0.312 when comparing 2007–2009 with 2019–2021. In line with this vocabulary drift, Figure 2 shows that tokenizers built on earlier slices suffer a degradation in compression efficiency when applied to more recent text. Across all three domains, we observe a consistent trend that the older the tokenizer is, the lower its bytes-per-token becomes on the most recent datasets. Vocabulary drift can also be observed from the segmentation behaviors. For example, the word "␣blockchain" is split into ["␣block", "ch", "ain"] by the tokenizer built on WMT 2007–2009, while the tokenizer built on WMT 2019–2021 preserves it as a single token.

Overall, all the results indicate that the tokenizers constructed on earlier data become increasingly misaligned with more recent data distributions.

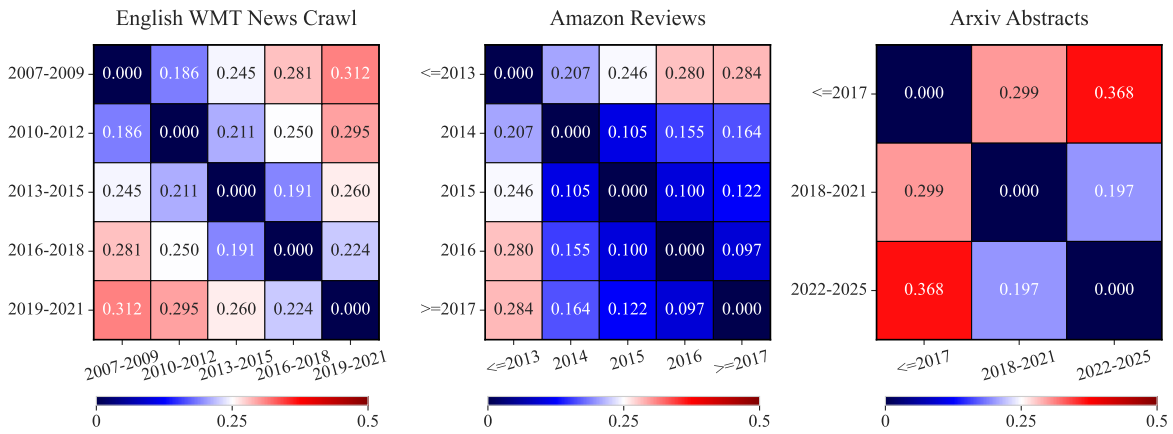

Figure 1: Jaccard-distance heatmaps comparing vocabularies of tokenizers built on different temporal slices across three datasets. Rows and columns denote the time spans used to build the corresponding vocabularies.

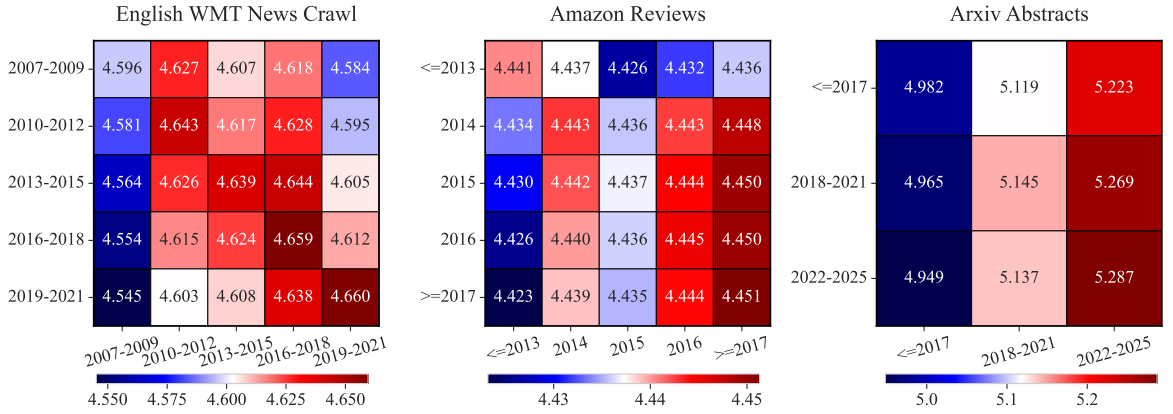

Figure 2: Bytes-per-token heatmaps for tokenizers evaluated across different temporal slices across three datasets. Rows correspond to the time span used to build the tokenizer, and columns to the time span of the evaluation data.

3.2 Impact on Model Performance

To verify whether the misalignment between a tokenizer and the data distribution leads to degradation in task performance, we design a synthetic experiment to isolate this effect.

Experimental setup. We construct two datasets with the same instance format (`A+B=C` or `A-B=C`) but different value ranges, so that the underlying data distribution shifts while the task remains unchanged. Specifically, the original ("old") dataset consists of 3.9K examples where $A, B \in [0, 50]$ and $C \in [0, 100]$, while the evolved ("new") dataset consists of 60.7K examples with $A, B \in [0, 200]$ and $C \in [0, 400]$, representing the shifted data distribution.

We build two tokenizers respectively on the old dataset and the new dataset. Both tokenizers are initialized with the same character set. We train two language models from scratch on the new dataset

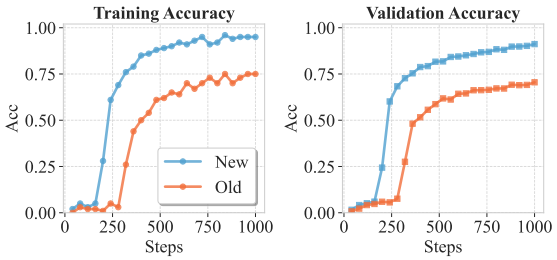

Figure 3: Training and validation accuracy on the synthetic task. The labels "Old" and "New" denote the trials trained with tokenizers built on the old and the new datasets, respectively.

under identical training settings. The only difference between the two trials lies in the tokenizer.

Findings. We observe the tokenizer that is misaligned with the data distribution leads to a substantial degradation in task performance. As shown in

Figure 3, the model trained with misaligned tokenizer (Old) performs worse than the model using the new tokenizer (New): training accuracy drops from 95.0% with the new tokenizer to 75.0% with the old tokenizer, validation accuracy drops from 91.2% to 70.5%. The misaligned tokenizer also yields poorer tokenization efficiency on the new data, with bytes-per-token decreasing from 1.37 to 1.22. These results highlight the impact of tokenizer misalignment on model performance, revealing the need for tokenizers that remain aligned with evolving data during lifelong learning.

4 Method

To develop an adaptive tokenizer for streaming data, the algorithm must (i) support vocabulary revision, including the deletion of out-of-date merge rules, and (ii) maintain up-to-date frequency estimates to guide the revision. We first adopt a graph-theoretic view of BPE and propose a simple yet effective mechanism that enables vocabulary revision and accurate, online frequency maintenance. The overall procedure of Ted-Tok is outlined in Algorithm 1.

4.1 Graph View of BPE and Sink Tokens

The merge list of a BPE tokenizer implicitly defines a directed acyclic graph (DAG) over the vocabulary $\mathcal{V}$. Formally, let the merge graph $G = (\mathcal{V}, E)$ denote this DAG, where each merge operation forming $w = \text{CONCAT}(u, v)$ induces edges (u, w) and (v, w) in E. Under this construction, only characters or special symbols w in the base vocabulary $\mathcal{V}_{\text{base}}$ have zero indegree, i.e., $\deg^-(w) = 0$.

Standard BPE iteratively merges pairs of adjacent tokens with the highest co-occurrence count to construct a vocabulary. As a result, any merged token w has a frequency less than that of its predecessors u and v. When considering the deletion of an outdated token from the vocabulary, for example, one with the lowest occurrence frequency in recent data, it follows directly that such a token must correspond to a sink node in the merge graph G. Another important constraint is that tokens in the base vocabulary $\mathcal{V}_{\text{base}}$ cannot be deleted, even if they are sink nodes. These tokens are essential for preserving complete coverage over characters and special symbols, which guarantees the tokenizer's ability to represent arbitrary input strings.

We next describe how Ted-Tok maintains the frequency estimates needed to reliably decide which sink token to remove and which token pair to merge on a data stream.

Algorithm 1 Our Proposed Ted-Tok Algorithm

Require: Data stream $\{\mathcal{D}_t\}_{t=1}^{T}$, base vocabulary $\mathcal{V}_{\text{base}}$, initial BPE tokenizer $(\mathcal{V}_1, \mathcal{M}_1)$, embedding weights $(\mathbf{E}, \mathbf{W}_{\text{out}})$, hyperparameters Δ_{revision}, T_{warmup}, β, α

Initialize $\mathcal{V}_1^{\text{sink}}$
Initialize frequency estimates $\tau_0^{\text{token}}, \tau_0^{\text{buffer}}, \tau_0^{\text{pair}}$ to 0
for $t \leftarrow 1$ **to** T **do**
    ... ▷ Update model weights via external training steps.
    Compute counts $c_t^{\text{token}}, c_t^{\text{pair}}$ from $\mathcal{D}_t$ given $\mathcal{V}_t$, $\mathcal{M}_t$
    Update frequency estimates $\tau_t^{\text{token}}, \tau_t^{\text{buffer}}, \tau_t^{\text{pair}}$ with c_t^{token} and c_t^{pair}
    if $t \geq T_{\text{warmup}}$ **and** $t \bmod \Delta_{\text{revision}} = 0$ **then**
        $(\mathcal{V}_{t+1}, \mathcal{M}_{t+1}, \mathcal{V}_{t+1}^{\text{sink}}, \tau_t^{\text{token}}, \tau_t^{\text{buffer}}, \tau_t^{\text{pair}}, \mathbf{E}, \mathbf{W}_{\text{out}})$ $\leftarrow$ VOCABULARYREVISION($\mathcal{V}_t, \mathcal{M}_t, \mathcal{V}_t^{\text{sink}}, \tau_t^{\text{token}}, \tau_t^{\text{buffer}}, \tau_t^{\text{pair}}, \mathcal{V}_{\text{base}}, \beta, \mathbf{E}, \mathbf{W}_{\text{out}}$)
    else
        $(\mathcal{V}_{t+1}, \mathcal{M}_{t+1}, \mathcal{V}_{t+1}^{\text{sink}}) \leftarrow (\mathcal{V}_t, \mathcal{M}_t, \mathcal{V}_t^{\text{sink}})$
    end if
end for

4.2 Frequency Maintenance on a Data Stream

Formally, let $\mathcal{V}_t$ and $\mathcal{M}_t$ denote the vocabulary and merge list at time step t. Ted-Tok primarily maintains two sets of statistics: the frequency of each token in $\mathcal{V}_t$ to identify outdated tokens, and the frequency of token pairs in $\mathcal{V}_t \times \mathcal{V}_t$ to identify candidate merge rules.

Given a stream of data arriving in batches $\mathcal{D}_{1:T}$, Ted-Tok performs frequency maintenance updates iteratively. At each time step t, we tokenize the batch $\mathcal{D}_t$ using $\mathcal{V}_t$ and $\mathcal{M}_t$. We then count the occurrences of each token $a \in \mathcal{V}_t$ in the tokenized $\mathcal{D}_t$, denoted by $c_t^{\text{token}}[a]$, and the number of times a token $u \in \mathcal{V}_t$ is immediately followed by a token $v \in \mathcal{V}_t$ in the tokenized sequences, denoted by $c_t^{\text{pair}}[u, v]$.

To stabilize the statistics over time, we apply an exponential moving average (EMA; Ruppert 1988) to these counts across time steps. For token pairs, $\tau_t^{\text{pair}}[u, v]$ tracks the averaged frequency of token pairs over time.

$$\tau_t^{\text{pair}}[u, v] = (1 - \alpha)\tau_{t-1}^{\text{pair}}[u, v] + \alpha c_t^{\text{pair}}[u, v]$$

$\alpha \in (0, 1)$ is the EMA smoothing factor determining the rate at which historical information decays.

For individual tokens, a naive approach is to track the historical frequency of each token and delete those with low frequency. However, we note that when a sink token w_{delete}, formed by merging u_{delete} and v_{delete} in $\mathcal{M}_t$, is removed, its predecessors u_{delete} and v_{delete}'s frequency must be recalibrated, since the frequency of w_{delete} is redistributed to them upon its deletion.

To manage this recalibration without recursive history scans, we maintain two distinct estimates for each token $a \in \mathcal{V}_t$: $\tau_t^{\text{token}}[a]$ and $\tau_t^{\text{buffer}}[a]$. Here, $\tau_t^{\text{token}}[a]$ is the standard EMA-based frequency estimate, while $\tau_t^{\text{buffer}}[a]$ serves as a buffer used for frequency recalibration. This buffer stores the frequency credits that are currently associated with a but must be redistributed to its predecessors when a is deleted.

At each time step t, both $\tau_t^{\text{token}}[a]$ and $\tau_t^{\text{buffer}}[a]$ are first updated via EMA based on the token count $c_t^{\text{token}}[a]$ in the current batch:

$$\tau_t^{\text{token}}[a] = (1-\alpha)\tau_{t-1}^{\text{token}}[a] + \alpha c_t^{\text{token}}[a]$$
$$\tau_t^{\text{buffer}}[a] = (1-\alpha)\tau_{t-1}^{\text{buffer}}[a] + \alpha c_t^{\text{token}}[a]$$

Both estimates will be updated identically until the vocabulary is changed. If we add a new token $w_{\text{new}} = \text{CONCAT}(u_{\text{new}}, v_{\text{new}})$ to the vocabulary at time t, we initialize $\tau_t^{\text{token}}[w_{\text{new}}]$ using $\tau_t^{\text{pair}}[u_{\text{new}}, v_{\text{new}}]$, while its buffer $\tau_t^{\text{buffer}}[w_{\text{new}}]$ starts at zero, as no historical credit has been deferred yet:

$$\tau_t^{\text{token}}[w_{\text{new}}] = \tau_t^{\text{pair}}[u_{\text{new}}, v_{\text{new}}]$$
$$\tau_t^{\text{buffer}}[w_{\text{new}}] = 0$$

If we remove a token $w_{\text{delete}} = \text{CONCAT}(u_{\text{delete}}, v_{\text{delete}})$ from the vocabulary at time t, the frequency in its buffer is redistributed to its predecessors:

$$\tau_t^{\text{token}}[u_{\text{delete}}] = \tau_t^{\text{token}}[u_{\text{delete}}] + \tau_t^{\text{buffer}}[w_{\text{delete}}]$$
$$\tau_t^{\text{buffer}}[u_{\text{delete}}] = \tau_t^{\text{buffer}}[u_{\text{delete}}] + \tau_t^{\text{buffer}}[w_{\text{delete}}]$$
$$\tau_t^{\text{token}}[v_{\text{delete}}] = \tau_t^{\text{token}}[v_{\text{delete}}] + \tau_t^{\text{buffer}}[w_{\text{delete}}]$$
$$\tau_t^{\text{buffer}}[v_{\text{delete}}] = \tau_t^{\text{buffer}}[v_{\text{delete}}] + \tau_t^{\text{buffer}}[w_{\text{delete}}]$$

The following proposition shows the correctness of this frequency redistribution process (see Appendix A for the proof):

Proposition 1 *The frequency redistribution mechanism preserves the statistical consistency of the estimators, such that the updated $\tau_t^{token}[u_{delete}]$ and $\tau_t^{token}[v_{delete}]$ accurately reflect the total frequency of u_{delete} and v_{delete} as if w_{delete} had been decomposed in all previous time steps.*

For ease of reference, we denote the set of frequency estimates at time t as $\mathcal{T}_t = \{\tau_t^{\text{token}}[\cdot], \tau_t^{\text{buffer}}[\cdot], \tau_t^{\text{pair}}[\cdot,\cdot]\}$.

Algorithm 2 Function VOCABULARYREVISION

Require: Current Ted-Tok state $(\mathcal{V}_t, \mathcal{M}_t, \mathcal{V}_t^{\text{sink}})$, frequency estimates $(\tau_t^{\text{token}}, \tau_t^{\text{buffer}}, \tau_t^{\text{pair}})$, base vocabulary $\mathcal{V}_{\text{base}}$, hyperparameters β, model weights $(\mathbf{E}, \mathbf{W}_{\text{out}})$

$(u_{\text{new}}, v_{\text{new}}) \leftarrow \arg\max_{(u,v)\in\mathcal{V}_t\times\mathcal{V}_t} \tau_t^{\text{pair}}[u_{\text{new}}, v_{\text{new}}]$
$w_{\text{delete}} \leftarrow \arg\min_{w\in\mathcal{V}_t^{\text{sink}}} \tau_t^{\text{token}}[w]$
if $\tau_t^{\text{pair}}[u_{\text{new}}, v_{\text{new}}] > \beta \cdot \tau_t^{\text{token}}[w_{\text{delete}}]$ **then**
  Construct a new token $w_{\text{new}} \leftarrow \text{CONCAT}(u_{\text{new}}, v_{\text{new}})$
  Retrieve $(u_{\text{delete}}, v_{\text{delete}}) \in \mathcal{M}_t$ such that $w_{\text{delete}} = \text{CONCAT}(u_{\text{delete}}, v_{\text{delete}})$
  Update $\mathcal{V}_{t+1}$: add w_{new}, delete w_{delete}
  Update $\mathcal{M}_{t+1}$: append $(u_{\text{new}}, v_{\text{new}})$, remove $(u_{\text{delete}}, v_{\text{delete}})$
  Update $\mathcal{V}_{t+1}^{\text{sink}}$: add w_{new}, delete $u_{\text{new}}, v_{\text{new}}$ if present, and add u_{delete} or v_{delete} if they become sink nodes
  Update frequency estimates at time t in-place: $\tau_t^{\text{token}}[w_{\text{new}}] \leftarrow \tau_t^{\text{pair}}[u_{\text{new}}, v_{\text{new}}]$, $\tau_t^{\text{buffer}}[w_{\text{new}}] \leftarrow 0$, redistribute $\tau_t^{\text{buffer}}[w_{\text{delete}}]$ to u_{delete} and v_{delete}, $\tau_t^{\text{pair}}[u_{\text{delete}}, v_{\text{delete}}] \leftarrow \tau_t^{\text{token}}[w_{\text{delete}}]$
  Initialize $\mathbf{E}[w_{\text{new}}], \mathbf{W}_{\text{out}}[w_{\text{new}}]$
end if
return $(\mathcal{V}_{t+1}, \mathcal{M}_{t+1}, \mathcal{V}_{t+1}^{\text{sink}}, \tau_t^{\text{token}}, \tau_t^{\text{buffer}}, \tau_t^{\text{pair}}, \mathbf{E}, \mathbf{W}_{\text{out}})$

4.3 Vocabulary Revision

At time step $t = 1$, Ted-Tok initializes from an existing tokenizer with vocabulary $\mathcal{V}_1$ and merge list $\mathcal{M}_1$. Correspondingly, the initial sink token set $\mathcal{V}_1^{\text{sink}}$ is constructed by identifying all tokens that do not appear as constituents of the merge rules in $\mathcal{M}_1$ and are not in the base vocabulary $\mathcal{V}_{\text{base}}$:

$$\mathcal{V}_1^{\text{sink}} = \mathcal{V}_1 \setminus \Big(\mathcal{V}_{\text{base}} \cup \bigcup_{(u,v)\in\mathcal{M}_1} \{u, v\}\Big)$$

This construction ensures that the sink token set $\mathcal{V}_1^{\text{sink}}$ contains exactly those tokens with zero out-degree and non-zero indegree in the merge graph, i.e., $\deg^+(w) = 0$ and $\deg^-(w) \neq 0$.

To ensure that vocabulary revisions are driven by reliable frequency estimates, we incorporate a warmup phase of T_{warmup} steps. During this interval $t \in [1, T_{\text{warmup}}]$, we only update the frequency estimates $\mathcal{T}_t$ from the data stream without executing any vocabulary revisions.

To balance efficiency and effectiveness, vocabulary revision is performed every Δ_{revision} steps, as outlined in Algorithm 2. At a vocabulary-revision step t, we aim to update $(\mathcal{V}_t, \mathcal{M}_t, \mathcal{V}_t^{\text{sink}})$ according to $\mathcal{T}_t$. For selecting candidate merge rules and new tokens, we choose the most frequent pair $(u_{\text{new}}, v_{\text{new}})$ according to $\tau_t^{\text{pair}}[\cdot,\cdot]$. For the token deletion candidate, we choose the least frequent token $w_{\text{delete}} = \text{CONCAT}(u_{\text{delete}}, v_{\text{delete}}) \in \mathcal{V}_t^{\text{sink}}$ according to $\tau_t^{\text{token}}[\cdot]$. To ensure consistency in vocabulary revision, we impose a constraint whereby

additions and deletions are triggered only when the token to be added is more "popular" than the one to be deleted:

$$\tau_t^{\text{pair}}[u_{\text{new}}, v_{\text{new}}] > \beta \tau_t^{\text{token}}[w_{\text{delete}}]$$

Here, $\beta > 1$ is a hyperparameter setting the margin.

Once the revision is triggered, we first update $(\mathcal{V}_t, \mathcal{M}_t, \mathcal{V}_t^{\text{sink}})$ by adding the new token $w_{\text{new}} = \text{CONCAT}(u_{\text{new}}, v_{\text{new}})$ as follows:

$$\begin{aligned}
\mathcal{V}_{t+1} &= \mathcal{V}_t \cup \{w_{\text{new}}\} \\
\mathcal{M}_{t+1} &= \mathcal{M}_t.\text{APPEND}((u_{\text{new}}, v_{\text{new}})) \\
\mathcal{V}_{t+1}^{\text{sink}} &= (\mathcal{V}_t^{\text{sink}} \cup \{w_{\text{new}}\}) \setminus \{u_{\text{new}}, v_{\text{new}}\}
\end{aligned}$$

Here, we directly add w_{new} into $\mathcal{V}_{t+1}^{\text{sink}}$, since w_{new} has no successors in the updated merge graph, while removing u_{new} and v_{new} from $\mathcal{V}_{t+1}^{\text{sink}}$, as they are no longer sink nodes. Then we update the frequency estimates $\tau_t^{\text{token}}[w_{\text{new}}]$ and $\tau_t^{\text{buffer}}[w_{\text{new}}]$ as mentioned in Section 4.2. As the token w_{new} is newly introduced, we initialize its embedding by averaging the embeddings of its predecessors u_{new} and v_{new}, ensuring a smooth transition, following the strategy employed by Gu et al. (2024); Li et al. (2025).

For deletion, we remove the selected sink token $w_{\text{delete}} = \text{CONCAT}(u_{\text{delete}}, v_{\text{delete}})$ as follows:

$$\begin{aligned}
\mathcal{V}_{t+1} &= \mathcal{V}_{t+1} \setminus \{w_{\text{delete}}\} \\
\mathcal{M}_{t+1} &= \mathcal{M}_{t+1}.\text{REMOVE}((u_{\text{delete}}, v_{\text{delete}})) \\
\mathcal{V}_{t+1}^{\text{sink}} &= (\mathcal{V}_{t+1}^{\text{sink}} \setminus \{w_{\text{delete}}\}) \cup \{a \in \{u_{\text{delete}}, v_{\text{delete}}\} \\
&\quad \mid \deg^+(a) = 0, \deg^-(a) \neq 0\}
\end{aligned}$$

Here, we remove w_{delete} from $\mathcal{V}_{t+1}^{\text{sink}}$, and check whether its predecessors u_{delete} and v_{delete} are new sink nodes. At the same time, we update the pair frequency $\tau_t^{\text{pair}}[u_{\text{delete}}, v_{\text{delete}}]$ and redistribute the deferred frequency credits from $\tau_t^{\text{buffer}}[w_{\text{delete}}]$ to u_{delete} and v_{delete}, as detailed in Section 4.2.

5 Experiment

In this section, we conduct experiments to validate the effectiveness of Ted-Tok.

5.1 Implementation

Simulating streaming data. We evaluate Ted-Tok on three tasks: a synthetic arithmetic (SA) task following the setting in Section 3.2; a machine translation (MT) task using WMT English–German datasets (2009–2019)[1]; and a question answering (QA) task using StreamingQA (Liška et al., 2022) covering English news articles from 2007 to 2021. For MT and QA, we simulate temporal drift by partitioning the corpora into yearly subsets. See Appendix B for details.

Training Settings. Across all tasks, data are organized according to temporal order. For SA, we follow the setup in Section 3.2 and define an "old" dataset and a "new" dataset, which are treated as two sequentially ordered subsets during training. For MT and QA, datasets are partitioned into yearly subsets to simulate temporal drift.

Based on this temporal data organization, we perform lifelong training that proceeds sequentially over temporally ordered data subsets using a Warmup–Stable–Decay learning-rate schedule. All models are evaluated on the test set corresponding to the latest period.

When using Ted-Tok, for SA, we set $T_{\text{warmup}}/T = 10\%$, $\Delta_{\text{revision}} = 300$, $\alpha = 0.3$, and $\beta = 1.2$, and disable vocabulary revision during the final 20% of training steps to ensure the embeddings of all newly added tokens are sufficiently trained. For MT and QA, we set $T_{\text{warmup}}/T = 3\%$, $\Delta_{\text{revision}} = 1000$, $\alpha = 5 \times 10^{-5}$, and $\beta = 1.2$, and disable revision during the final 30% of steps. Additionally, for these two tasks, we batch 100 token additions and deletions per revision step to mitigate the overhead associated with reloading the tokenizer.

More implementation details are provided in Appendix C.

Baselines. We evaluate three tokenizer configurations within the lifelong training scenario presented in Table 1: (1) *Old* uses a fixed tokenizer built on the earliest data subset, representing a setting where the vocabulary remains static despite temporal drift. (2) *New* involves an abrupt transition where the original tokenizer is replaced by a new one built on the latest data subset before training on that subset. (3) *Ted-Tok* employs our method to evolve the initial vocabulary to incorporate emerging linguistic patterns from the latest data subset. All three settings follow an identical lifelong training procedure on all data subsets prior to the latest data subset.

Evaluation Metrics. To evaluate overall performance, we report Exact Match for the synthetic task, BLEU (Papineni et al., 2002) for translation quality, and Contains Accuracy for QA (measuring whether the generation contains the gold answer). To evaluate tokenization quality, we report Bytes

[1] https://www.statmt.org/

Task	Metric	Tokenizer	Performance (↑)
SA	Exact Match	Old	82.30%
		New	88.15%
		Ted-Tok	**90.55%**
MT	BLEU	Old	34.9125
		New	35.3654
		Ted-Tok	**35.9447**
QA	Contains Accuracy	Old	31.52%
		New	32.17%
		Ted-Tok	**35.36%**

Table 1: Quantitative comparison on different downstream tasks. (↑) indicates higher is better. The best results are highlighted in **bold**.

Task	SA	MT	QA
Jaccard Distance (↓)			
Old vs. New	0.7013	0.6185	0.3123
Ted-Tok vs. New	0.2456	0.3855	0.1229
Bytes Per Token (↑)			
New (Reference)	1.366	4.057	4.660
Old	1.216	3.655	4.584
Ted-Tok	1.359	4.034	4.655

Table 2: Tokenizer metrics comparison across tasks. *Old* and *New* denote tokenizers built on the earliest and latest subsets, respectively. *Ted-Tok* represents the evolved Ted-Tok vocabulary after adapting to the latest data subset.

Per Token for compression efficiency and the Jaccard Distance relative to "New Tok" as a measure of misalignment with the current data.

5.2 Results and Analysis

First, regarding the overall task performance, Table 1 demonstrates that Ted-Tok significantly outperforms the *New* configuration, even though the latter already provides marginal gains over the *Old* configuration. The results indicate that evolving the established vocabulary is more effective than a complete reset. While the *New* configuration captures recent linguistic shifts, it lacks continuity with previously learned representations, leading to a loss of prior knowledge. In contrast, Ted-Tok provides a more seamless evolutionary path for the vocabulary, balancing the preservation of prior knowledge with the need to adapt to vocabulary drift.

To further investigate how Ted-Tok behaves during adaptation, we compare the evolved Ted-Tok with the original fixed tokenizer and the new tokenizer in terms of vocabulary overlap and compression efficiency. As shown in Table 2, two key observations emerge. First, Ted-Tok significantly reduces the Jaccard Distance to the new tokenizer

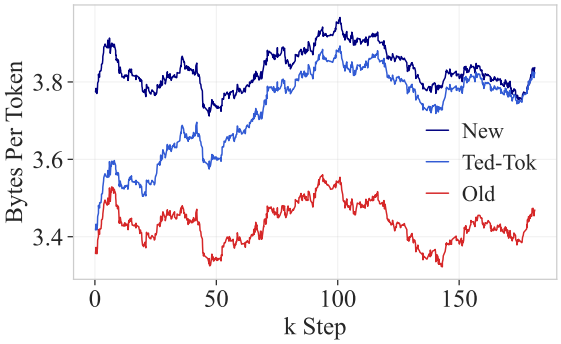

Figure 4: Training-time bytes per token (smoothed with a 256-step window).

compared to the old one. This indicates that Ted-Tok effectively mitigates tokenizer-data misalignment caused by temporal drift. Second, regarding the Bytes Per Token metric, Ted-Tok demonstrates consistently higher compression efficiency than the old tokenizer, achieving a level comparable to the new tokenizer. We further examine the dynamic progression of this adaptation in the Machine Translation task. As illustrated in Figure 4, Ted-Tok gradually improves its compression efficiency throughout the adaptation process, eventually converging to the performance of the new tokenizer. This gradual refinement underscores Ted-Tok's ability to incrementally incorporate emerging linguistic patterns.

Overall, these results show that improving tokenizer–data alignment via Ted-Tok yields higher compression efficiency and better downstream performance in lifelong learning.

5.3 Ablation study

We conduct ablation analyses on key design choices in Ted-Tok: (1) Δ_{revision}, the intensity of vocabulary updates, specifically the revision interval, (2) T_{warmup}, the warmup phase duration, (3) α, the EMA smoothing factor, and (4) β, the margin coefficient. We perform these ablation studies on the Machine Translation task and evaluate them using the BLEU score. The results are illustrated in Table 3.

Revision Interval Δ_{revision}. When evaluating the revision interval Δ_{revision}, we find that setting $\Delta_{\text{revision}} = 1000$ yields optimal performance. Performance degrades at both extremes of the interval, indicating a critical trade-off: overly sparse updates result in insufficient adaptation, while excessively frequent revisions disrupt embedding convergence.

Warmup Phase Duration T_{warmup}. By compar-

Hyperparameter Change	BLEU (↑)	Bytes Per Token (↑)
Ours	35.9447	4.034
$\Delta_{\text{revision}} = 500$	35.7798	4.071
$\Delta_{\text{revision}} = 1500$	35.5559	3.989
$\Delta_{\text{revision}} = 2000$	35.0167	3.957
$T_{\text{warmup}}/T = 0.01$	35.4943	4.037
$T_{\text{warmup}}/T = 0.02$	35.4249	4.035
$\alpha = 5 \times 10^{-4}$	36.0244	4.039
$\alpha = 5 \times 10^{-6}$	35.7474	4.032
$\beta = 1.6$	35.8615	4.048
$\beta = 2.0$	35.7366	4.048

Table 3: Ablation study on hyperparameter variations. *Ours* uses Δ_{revision}=1000, T_{warmup}/T=0.03, $\beta = 1.2$ and $\alpha = 5 \times 10^{-5}$. Each row varies a single hyperparameter while keeping all others fixed.

ing different values of T_{warmup}/T, we identify that a 3% warmup phase is essential for guaranteeing the reliability of frequency estimates. While a negligible warmup enables earlier adaptation, it leads to "noisy" revisions based on inaccurate statistics. Conversely, an extended warmup unnecessarily delays the optimization of the vocabulary.

Smoothing Factor α and Margin Coefficient β. Experiments across varying smoothing factor (α) and margin coefficient (β) yield similar performance, demonstrating the method's robustness on these hyperparameters.

5.4 Computational Overhead.

Tokenizer adaptation incurs negligible overhead. On the Machine Translation task, training on the latest data subset with a fixed tokenizer requires 624.0 GPU hours, while tokenizer adaptation adds only 8.3 CPU hours for frequency maintenance and vocabulary revision.

6 Related Work

Effects of Tokenization in Language Models. Tokenizers play a crucial role in language models and have been studied from several perspectives, including information-theoretic measures of tokenization efficiency (Zouhar et al., 2023), inductive biases in numerical reasoning induced by number tokenization (Singh and Strouse, 2024), robustness to adversarial inputs that exploit token segmentation failures (Wang et al., 2024), and tokenization effects on token sampling (Phan et al., 2025). However, these studies largely focus on static settings and rarely examine how tokenizers should adapt under temporal drift in lifelong learning.

Lifelong Learning. Lifelong learning aims to handle temporal drift in language and data distributions, which can cause models trained on earlier data to become outdated and misaligned with current usage (Shi et al., 2024; Sun et al., 2020; Kutuzov et al., 2018). Most prior work on temporal adaptation operates at the level of model weights, either by updating parameters via continual pretraining on time-ordered data (Jin et al., 2022; Jang et al., 2022; Su et al., 2023; Qin et al., 2023) or by injecting temporal signals such as timestamps or temporal attention to encode time-aware representations (Dhingra et al., 2022; Rosin et al., 2022). However, these approaches typically assume a static tokenizer, which is problematic because as token usage shifts over time, the mismatch between newer data and a fixed tokenizer can grow, reducing tokenization efficiency and ultimately degrading task performance.

Dynamic Vocabulary and Tokenizer Adaptation. Prior work on adaptive vocabularies and tokenizer adaptation mainly targets adaptation to domain or language shifts (Liu et al., 2023; Balde et al., 2024; Sharthak et al., 2025), improving multilingual fairness (Ahia et al., 2024), or tailoring the vocabulary to the model architecture (Zheng et al., 2024). However, these approaches typically assume adaptation to static corpora and thus do not readily extend to lifelong learning settings. Beyond these, Amba Hombaiah et al. (2021) also consider vocabulary updates under temporal drift, but their approach assumes the presence of explicit hashtag signals in the data and is tailored to a Twitter-specific BERT-based model. To the best of our knowledge, we are the first to introduce an adaptive tokenizer for lifelong learning under temporal drift that makes no domain-specific assumptions about the underlying data stream.

7 Conclusion

We propose the Temporal Drift Tokenizer (Ted-Tok), an adaptive BPE tokenizer designed for lifelong learning that maintains a flexible vocabulary evolving with the data stream. Across synthetic and real-world tasks, Ted-Tok consistently mitigates vocabulary misalignment caused by temporal drift, thereby preserving high compression efficiency and improving task performance in lifelong learning. Our findings validate that adaptive tokenization is an essential and promising paradigm for lifelong learning.

8 Limitations

While our work demonstrates the effectiveness of Ted-Tok, several limitations suggest directions for future research. Firstly, the current design of Ted-Tok is based on BPE, and adapting it to alternative schemes such as Unigram (Kudo, 2018) remains unexplored. Secondly, we use a naive embedding initialization strategy for newly added tokens, which could be improved with more advanced methods. In addition, further exploration is required to discover how fixed tokenizers limit the lifelong learning capabilities of larger language models and the corresponding benefits that adaptive tokenization could offer as model scale increases. Moreover, our approach may be misused in ways that exacerbate unfairness toward low-resource languages, which would require additional considerations in practical applications.

References

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Valentin Hofmann, Tomasz Limisiewicz, Yulia Tsvetkov, and Noah A Smith. 2024. Magnet: Improving the multilingual fairness of language models with adaptive gradient-based tokenization. In *Advances in Neural Information Processing Systems*, volume 37, pages 47790–47814. Curran Associates, Inc.

Spurthi Amba Hombaiah, Tao Chen, Mingyang Zhang, Michael Bendersky, and Marc Najork. 2021. Dynamic language models for continuously evolving content. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2514–2524.

Jason Ansel, Edward 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, and 30 others. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM.

Gunjan Balde, Soumyadeep Roy, Mainack Mondal, and Niloy Ganguly. 2024. Adaptive BPE tokenization for enhanced vocabulary adaptation in finetuning pretrained language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 14724–14733, Miami, Florida, USA. Association for Computational Linguistics.

Loïc Barrault, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, Christof Monz, Mathias Müller, Santanu Pal, Matt Post, and Marcos Zampieri. 2019. Findings of the 2019 conference on machine translation (WMT19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61, Florence, Italy. Association for Computational Linguistics.

Ondřej Bojar, Christian Buck, Chris Callison-Burch, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia. 2013. Findings of the 2013 Workshop on Statistical Machine Translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 1–44, Sofia, Bulgaria. Association for Computational Linguistics.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Raphael Rubino, Lucia Specia, and Marco Turchi. 2017. Findings of the 2017 conference on machine translation (WMT17). In *Proceedings of the Second Conference on Machine Translation*, pages 169–214, Copenhagen, Denmark. Association for Computational Linguistics.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, and 2 others. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Barry Haddow, Matthias Huck, Chris Hokamp, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Carolina Scarton, Lucia Specia, and Marco Turchi. 2015. Findings of the 2015 workshop on statistical machine translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 1–46, Lisbon, Portugal. Association for Computational Linguistics.

Ondřej Bojar, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, and Christof Monz. 2018. Findings of the 2018 conference on machine translation (WMT18).

In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 272–303, Belgium, Brussels. Association for Computational Linguistics.

Chris Callison-Burch, Philipp Koehn, Christof Monz, Kay Peterson, Mark Przybocki, and Omar Zaidan. 2010. Findings of the 2010 joint workshop on statistical machine translation and metrics for machine translation. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 17–53, Uppsala, Sweden. Association for Computational Linguistics.

Chris Callison-Burch, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia. 2012. Findings of the 2012 workshop on statistical machine translation. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 10–51, Montréal, Canada. Association for Computational Linguistics.

Chris Callison-Burch, Philipp Koehn, Christof Monz, and Josh Schroeder. 2009. Findings of the 2009 Workshop on Statistical Machine Translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 1–28, Athens, Greece. Association for Computational Linguistics.

Chris Callison-Burch, Philipp Koehn, Christof Monz, and Omar Zaidan. 2011. Findings of the 2011 workshop on statistical machine translation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 22–64, Edinburgh, Scotland. Association for Computational Linguistics.

Iaroslav Chelombitko, Egor Safronov, and Aleksey Komissarov. 2024. Qtok: A comprehensive framework for evaluating multilingual tokenizer quality in large language models. *CoRR*, abs/2410.12989.

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.

Shuhao Gu, Mengdi Zhao, Bowen Zhang, Liangdong Wang, Jijie Li, and Guang Liu. 2024. Retok: Replacing tokenizer to enhance representation efficiency in large language model. *Preprint*, arXiv:2410.04335.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, and 1 others. 2024. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*.

Joel Jang, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, Stanley Jungkyu Choi, and Minjoon Seo. 2022. Towards continual knowledge learning of language models. *Preprint*, arXiv:2110.03215.

Xisen Jin, Dejiao Zhang, Henghui Zhu, Wei Xiao, Shang-Wen Li, Xiaokai Wei, Andrew Arnold, and Xiang Ren. 2022. Lifelong pretraining: Continually adapting language models to emerging corpora. In *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, pages 1–16, virtual+Dublin. Association for Computational Linguistics.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Andrey Kutuzov, Lilja Øvrelid, Terrence Szymanski, and Erik Velldal. 2018. Diachronic word embeddings and semantic shifts: a survey. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1384–1397, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Chenghao Li, Liu Liu, Baosheng Yu, Jiayan Qiu, and Yibing Zhan. 2025. Re-initialization token learning for tool-augmented large language models. *ArXiv*, abs/2506.14248.

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*.

Siyang Liu, Naihao Deng, Sahand Sabour, Yilin Jia, Minlie Huang, and Rada Mihalcea. 2023. Task-adaptive tokenization: Enhancing long-form text generation efficacy in mental health and beyond. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15264–15281, Singapore. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

OpenAI. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Buu Phan, Brandon Amos, Itai Gat, Marton Havasi, Matthew J. Muckley, and Karen Ullrich. 2025. Exact byte-level probabilities from tokenized language models for FIM-tasks and model ensembles. In

The Thirteenth International Conference on Learning Representations.

Yujia Qin, Cheng Qian, Xu Han, Yankai Lin, Huadong Wang, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2023. Recyclable tuning for continual pre-training. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 11403–11426, Toronto, Canada. Association for Computational Linguistics.

Guy D. Rosin, Ido Guy, and Kira Radinsky. 2022. Time masking for temporal language models. In *WSDM '22: The Fifteenth ACM International Conference on Web Search and Data Mining, Virtual Event / Tempe, AZ, USA, February 21 - 25, 2022*, pages 833–841. ACM.

David Ruppert. 1988. Efficient estimations from a slowly convergent robbins-monro process. Technical Report 781, Cornell University Operations Research and Industrial Engineering. Previously published as: 781.

Tomohiro Sawada and Kartik Goyal. 2025. Train it and forget it: Merge lists are unnecessary for BPE inference in language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 35033–35046, Suzhou, China. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Shaurya Sharthak, Vinayak Pahalwan, Adithya Kamath, and Adarsh Shirawalmath. 2025. Achieving tokenizer flexibility in language models through heuristic adaptation and supertoken learning. *arXiv preprint arXiv:2505.09738*.

Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. 2024. Continual learning of large language models: A comprehensive survey. *Preprint*, arXiv:2404.16789.

Aaditya K. Singh and DJ Strouse. 2024. Tokenization counts: the impact of tokenization on arithmetic in frontier llms. *Preprint*, arXiv:2402.14903.

Zhaochen Su, Juntao Li, Zikang Zhang, Zihan Zhou, and Min Zhang. 2023. Efficient continue training of temporal language model with structural information. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6315–6329, Singapore. Association for Computational Linguistics.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. 2020. Ernie 2.0: A continual pre-training framework for language understanding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8968–8975.

Lintang Sutawika, Aran Komatsuzaki, and Colin Raffel. 2024. Pile-t5. Blog post.

Dixuan Wang, Yanda Li, Junyuan Jiang, Zepeng Ding, Guochao Jiang, Jiaqing Liang, and Deqing Yang. 2024. Tokenization matters! degrading large language models through challenging their tokenization. *CoRR*, abs/2405.17067.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Mengyu Zheng, Hanting Chen, Tianyu Guo, Chong Zhu, Binfan Zheng, Chang Xu, and Yunhe Wang. 2024. Enhancing large language models through adaptive tokenizers. In *Advances in Neural Information Processing Systems*, volume 37, pages 113545–113568. Curran Associates, Inc.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Mrinmaya Sachan, and Ryan Cotterell. 2023. Tokenization and the noiseless channel. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5184–5207, Toronto, Canada. Association for Computational Linguistics.

A Proof of proposition 1

Proof 1 *Let $t_{start} \in \mathbb{Z}$ denote the time step when token $w = \text{CONCAT}(u, v)$ was added to the vocabulary. Let $t \in \mathbb{Z}$ denote the current time step when w is selected to be deleted. When w was not in the vocabulary (i.e. $l \leq t_{start}$), the occurrences of the sequence (u, v) were tokenized as separate tokens u and v.*

For the period $l \in (t_{start}, t]$, adjacent occurrences of u and v are merged into w, causing $c_l^{token}[u]$ and $c_l^{token}[v]$ fail to capture these occurrences. Since the buffer $\tau_t^{buffer}[w]$ is initialized to 0 at t_{start}, $\tau_t^{buffer}[w]$ accumulates exactly these missed counts during the active lifespan of w:

$$\tau_t^{buffer}[w] = \sum_{l=t_{start}+1}^{t} \alpha(1-\alpha)^{t-l} c_l^{token}[w]. \quad (1)$$

When deleting w, redistributing this count is equivalent to decomposing all occurrences of w back into u and v for the period $(t_{start}, t]$:

$$c_l^{token'}[u] = c_l^{token}[u] + c_l^{token}[w], \quad \forall l \in (t_{start}, t]. \quad (2)$$

The corrected EMA estimate for u then becomes:

$$\begin{aligned}\tau_t^{token'}[u] &= \sum_{l=1}^{t_{start}} \alpha(1-\alpha)^{t-l} c_l^{token}[u] \\ &+ \sum_{l=t_{start}+1}^{t} \alpha(1-\alpha)^{t-l}(c_l^{token}[u] + c_l^{token}[w]) \\ &= \sum_{l=1}^{t} \alpha(1-\alpha)^{t-l} c_l^{token}[u] \\ &+ \sum_{l=t_{start}+1}^{t} \alpha(1-\alpha)^{t-l} c_l^{token}[w] \\ &= \tau_t^{token}[u] + \tau_t^{buffer}[w].\end{aligned}$$

A symmetric derivation holds for v. Thus, the redistribution yields frequency estimates $\tau_t^{token'}[u]$, $\tau_t^{token'}[v]$ mathematically equivalent to those obtained if w had effectively been decomposed into u and v throughout the entire data stream.

B Dataset statistics

Synthetic Arithmetic Dataset. Both datasets are generated to ensure balanced coverage of addition and subtraction operations. Each instance follows the format `A+B=C` or `A-B=C`. A 2K subset from Dataset B is reserved as the validation set for all experiments. Table 4 presents the dataset statistics.

WMT is released by the *Workshop on Statistical Machine Translation* (Callison-Burch et al., 2009, 2010, 2011, 2012; Bojar et al., 2013, 2014, 2015, 2016, 2017, 2018; Barrault et al., 2019), providing large-scale parallel corpora from diverse sources such as News Commentary, Europarl, and Common Crawl. We process the WMT English–German parallel corpus following the standard WMT preprocessing and tokenization pipeline to ensure comparability with prior work. Table 5 summarizes the size of each released version of the component corpus, and Table 6 maps yearly releases (2009–2019) to their included sources.

StreamingQA. The StreamingQA dataset is derived from the English WMT News Crawl corpus spanning 2007–2021 and is publicly available on GitHub[2]. It is constructed by aligning news articles with temporally grounded question–answer (QA) annotations, enabling the evaluation of how models adapt to evolving information over time.

For lifelong learning setting, we group the yearly training data into consecutive three-year intervals. Table 7 provides the training data statistics.

[2] https://github.com/google-deepmind/streamingqa

Dataset	Range of Numbers	Number of Samples
Dataset A	0–50	3.9K
Dataset B	0–200	60.7K (2K for validation)

Table 4: Summary of Synthetic task Datasets.

Data Source	Version	#Sentence Pairs
News Commentary	v4 (2009)	83K
News Commentary	v5 (2010)	100K
News Commentary	v6 (2011)	136K
News Commentary	v7 (2012)	159K
News Commentary	v8 (2013)	178K
News Commentary	v9 (2014)	201K
News Commentary	v10 (2015)	216K
News Commentary	v11 (2016)	243K
News Commentary	v12 (2017)	271K
News Commentary	v13 (2018)	284K
News Commentary	v14 (2019)	338K
Europarl	v4	1.4M
Europarl	v5	1.5M
Europarl	v6	1.7M
Europarl	v7	1.9M
Europarl	v9	1.8M
Common Crawl	–	2.4M
ParaCrawl	2018[3]	36M
ParaCrawl	2019	94M
Rapid Corpus	–	1.3M
Wiki Titles	v1	1.3M

Table 5: Data sources and their version-level sizes in the WMT English–German parallel corpus.

To construct a reliable evaluation set for base-model assessment, we first use GPT-4o (OpenAI, 2024) to standardize the phrasing and convert each fact-based question and its corresponding answer from the 2019–2021 portion of the StreamingQA dataset into a consistent end-of-sequence completion format. We then use pile-t5-large (Sutawika et al., 2024) as the baseline model and apply a filtering step in which the baseline must produce the correct answer in eight independent deterministic decoding attempts. Items that fail this criterion are removed, resulting in a curated set of 1,660 questions. This question set is used in Section 5 for evaluating the QA task.

During evaluation, we generate five independent sampled completions for each question using a decoding configuration of temperature 0.2 and top-p 0.95. We report the mean performance across samples along with the sample variance.

Year	News Comm.	Europarl	Common Crawl	ParaCrawl	Rapid	Wiki Titles
2009	v4	v4	–	–	–	–
2010	v5	v5	–	–	–	–
2011	v6	v6	–	–	–	–
2012	v7	v7	–	–	–	–
2013	v8	v7	✓	–	–	–
2014	v9	v7	✓	–	–	–
2015	v10	v7	✓	–	–	–
2016	v11	v7	✓	–	–	–
2017	v12	v7	✓	–	✓	–
2018	v13	v7	✓	2018	✓	–
2019	v14	v9	✓	2019	✓	✓

Table 6: Yearly composition of the WMT En–De corpus (2009–2019). "✓" denotes the inclusion of that source in a given year.

Year Range	# Training Examples
2007–2009	6.08M
2010–2012	7.20M
2013–2015	12.49M
2016–2018	10.76M
2019–2021	15.88M

Table 7: Statistical Summary of the QA Training Dataset.

C Implementation Details

All training are implemented on PyTorch(Ansel et al., 2024) framework. Our tokenizer implementation is compatible with the `GPT2TokenizerFast` class provided by the TRANSFORMERS library (Wolf et al., 2020).

C.1 Synthetic Task

For synthetic arithmetic task, we use a two-layer, four-head Transformer model with ~4K parameters. For tokenizers, we restrict the vocabulary size to 50.

We train on a single NVIDIA RTX 4090 GPU and use a global batch size of 4096. The optimizer is Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$. We train for 1000 epochs for each data subsets. Both models are initialized with random embeddings and trained under matched hyperparameters. All settings adopt a constant learning rate of 5×10^{-4}.

When using Ted-Tok, we set $T_{\text{warmup}}/T = 10\%$, $\Delta_{\text{revision}} = 300$, $\alpha = 0.3$, and $\beta = 1.2$, and disable vocabulary revision during the final 20% of training steps to ensure the embeddings of all newly added tokens are sufficiently trained.

C.2 Real-world Tasks

In real-world tasks, we adopt the GPT-2 124M configuration, which includes a hidden size of 768, 12 transformer layers, 12 attention heads, and a context window of 1024 tokens. The same architecture is used for all baselines to ensure a controlled comparison. For tokenizers, we restrict the vocabulary size to 50257.

Multi-GPU training is performed with Distributed Data Parallel on 8 A100 GPUs. We use a global batch size of 576. The optimizer is AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and a weight decay of 0.01. When using Ted-Tok, we set $T_{\text{warmup}}/T = 3\%$, $\Delta_{\text{revision}} = 1000$, $\alpha = 5 \times 10^{-5}$, and $\beta = 1.2$, and disable revision during the final 30% of steps. Additionally, for these two tasks, we batch 100 token additions and deletions per revision step to mitigate the overhead associated with reloading the tokenizer.

For translation, the dataset is split by year, and the model is trained for three epochs on each yearly subset. For QA, the data are grouped into consecutive three-year intervals, and the model is trained for ten epochs on each interval. Both the baseline and Ted-Tok setting employ the same Warmup–Stable–Decay learning-rate schedule (Hu et al., 2024) with decayed peak learning rates (Loshchilov and Hutter, 2016).

For the first year-interval dataset, we set the peak learning rate to 6×10^{-4}. For each subsequent interval, the peak learning rate is decayed to $0.85\times$ the peak value used in the previous interval. During

intermediate year intervals, the cosine schedule is not allowed to decay to zero; instead, the minimum learning rate is clipped at 6×10^{-5}. Only on the final year interval does the cosine schedule decay fully toward zero.