

# Training Language Models with Memory Augmentation

Zexuan Zhong<sup>†</sup> Tao Lei<sup>\*</sup> Danqi Chen<sup>†</sup>

<sup>†</sup>Princeton University

{zzhong, danqi}@cs.princeton.edu, taole@google.com

## Abstract

Recent work has improved language models (LMs) remarkably by equipping them with a non-parametric memory component. However, most existing approaches only introduce memories at testing time or represent them using a separately trained encoder, resulting in sub-optimal training of the language model. In this work, we present TRIME, a novel yet simple training approach designed for training LMs with memory augmentation. Our approach uses a training objective that directly takes in-batch examples as accessible memory. We also present new methods for memory construction and data batching, which are used for adapting to different sets of memories—local, long-term, and external memory—at testing time. We evaluate TRIME on multiple language modeling and machine translation benchmarks and show that it is able to achieve significant improvements across all the settings. Concretely, TRIME reduces the perplexity from 18.70 to 15.37 on WIKITEXT-103, by effectively leveraging a large memory set from the training corpus. Compared to standard LM training, TRIME adds negligible computational overhead and is compatible with different neural architectures, making it a versatile solution for training memory-augmented LMs.<sup>1</sup>

## 1 Introduction

Memory augmentation has become a remarkable approach to enhance language modeling performance without significantly increasing the amount of parameters and computation. By accessing memory units such as a neural cache of recent inputs (Merity et al., 2017; Grave et al., 2017b) and an external look-up table (Khandelwal et al., 2020), a memory-augmented language model (LM) enjoys increased memorization capacity and sets

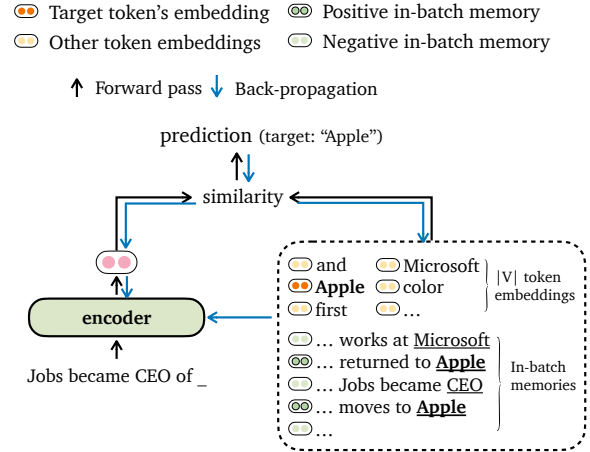


Figure 1: An illustration of our training objective. Our objective aligns the hidden representation with both token embeddings and a set of in-batch contextualized representations that are constructed during training.

new state-of-the-art records in various language modeling benchmarks.

A major limitation of existing approaches, however, is that the memory units are either introduced at *testing time* (Grave et al., 2017b,a; Khandelwal et al., 2020) or taken from a *separately trained model* (Yogatama et al., 2021). As a consequence, they are not directly optimized during the training process, resulting in a missed opportunity to achieve even stronger results. In this paper, we pioneer and present a novel yet simple training approach TRIME (Training with In-batch Memories)<sup>2</sup>, that is well-suited for memory augmentation in language modeling. Our approach makes two major departures compared to standard language model training:

**Training objective** Inspired by contrastive representation learning, we propose a training objective that directly leverages in-batch examples as accessible memory (Figure 1). Our training ob-

<sup>\*</sup>TL currently works at Google Research. The collaboration was initialized before TL joined Google.

<sup>1</sup>Our code and pre-trained models are publicly available at <https://github.com/princeton-nlp/TRIME>.

<sup>2</sup>We can also interpret TRIME as *three types of memories*, as we will elaborate in the paper.

jective is closely connected to neural cache models (Grave et al., 2017b; Merity et al., 2017) and nearest-neighbor language models (Khandelwal et al., 2020), where the next-token probabilities are calculated by comparing encoder outputs against static token embeddings *and* memory representations. However, previous work only considers incorporating memories at testing time, while we do for both training and testing.

**In-batch memory construction** With this training objective in mind, the key challenge is how to construct memories effectively *during training* while keeping it efficient. We identify three types of memories that can be leveraged at testing time and have been explored in the literature: (a) *local* memory denotes the words that appear in the recent past and are modeled using attention (Vaswani et al., 2017); (b) *long-term* memory<sup>3</sup> denotes long-range context from the same document but cannot be directly accessed due to the limit of input length; (c) *external* memory is used to store the entire training set or any additional corpus (Khandelwal et al., 2020; Borgeaud et al., 2021).

To better leverage these memories at testing time, we devise new *data batching* strategies to improve the construction of training memories (§4). By packing consecutive segments from the same document in one training batch, our model can access long-term memories beyond the attention context. We pack segments from other documents that have high lexical overlap as a proxy to all external memory units. Importantly, these working memories are generated on the fly during training, allowing us to back-propagate to all memory representations.

We instantiate TRIME in three models by considering different sets of training and testing memories (Table 1) and evaluate them on multiple language modeling and machine translation benchmarks. We highlight our results as follows:

- We first show that we can simply optimize a language model using our training objective *without* long-term and external memory. Without any other modifications, we demonstrate that a 247M Transformer-based model can achieve an improved perplexity from 18.70 to 17.76 on WIKITEXT-103 (Merity et al., 2017) with negligible overhead. This model can be viewed as a simple replacement

<sup>3</sup>Long-term memory may have different interpretations in other contexts and we use *long-term* memory to refer to long-range context in modeling long sequences, following previous work (Martins et al., 2022; Wu et al., 2022).

	Training Memory	Testing Memory
vanilla LM	None	None
cont. cache	None	$\mathcal{M}_{\text{local}}$ or $\mathcal{M}_{\text{long}}$
kNN-LM	None	$\mathcal{M}_{\text{ext}}$
TRIMELM	$\mathcal{M}_{\text{local}}$	$\mathcal{M}_{\text{local}}$
TRIMELM <sub>long</sub>	§4.2	$\mathcal{M}_{\text{local}}, \mathcal{M}_{\text{long}}$
TRIMELM <sub>ext</sub>	§4.3	$\mathcal{M}_{\text{local}}, \mathcal{M}_{\text{long}}, \mathcal{M}_{\text{ext}}$

Table 1: A comparison between our TRIME language models and previous approaches: vanilla LM, continuous cache (Grave et al., 2017b,a), kNN-LM (Khandelwal et al., 2020).  $\mathcal{M}_{\text{local}}, \mathcal{M}_{\text{long}}, \mathcal{M}_{\text{ext}}$  denote local, long-term and external memories respectively (§2.2).

for vanilla language models.

- By training with consecutive segments in the same batch, our approach is capable of leveraging very *long context* at testing time—up to 15k-25k tokens on WIKITEXT-103 and ENWIK8 (Mahoney, 2009). Our approach achieves at least competitive performance as previous works (Dai et al., 2019; Martins et al., 2022; Ji et al., 2022) that modify the Transformer architecture to incorporate memories from previous segments, yet our solution is conceptually simpler and computationally cheaper.

- Finally, we train language models by incorporating all other segments in the same batch as memories. Our model works better with a *large datastore* at testing time and improves over the kNN-LM model (Khandelwal et al., 2020) by reducing the test perplexity from 16.23 to 15.41 on WIKITEXT-103. We also demonstrate significant improvements over the kNN-MT baseline (Khandelwal et al., 2021) on an IWSLT’14 De-En machine translation task.

In summary, we propose a simple approach TRIME for optimizing language models with memory augmentation and demonstrate consistent and significant gains in multiple experimental settings. Our approach only uses memories at the final prediction step, and hence adds little computational overhead and can be combined with different model architectures such as recurrent networks and other attention variants (Lei, 2021; Dai et al., 2019; Rae et al., 2020). We hope that our work can encourage the research community to think about better training objectives for language models, given their significant societal impacts (Brown et al., 2020; Chowdhery et al., 2022; Zhang et al., 2022).

## 2 Preliminaries

### 2.1 Language Modeling

In this paper, we mainly focus on improving language models, although our solutions may extend to most text generation tasks (see one example of machine translation in §5.4). Neural language models take a sequence of tokens as context  $c_t = x_1, \dots, x_{t-1}$  and map it to a vector representation  $f_\theta(c_t) \in \mathbb{R}^d$ , where  $f_\theta(\cdot)$  is parameterized by a neural network. The next-token probability is:

$$P(w | c_t) \propto \exp(E_w^\top f_\theta(c_t)), \quad (1)$$

where  $E_w \in \mathbb{R}^d$  denotes the output embedding of token  $w \in \mathcal{V}$ . The parameters are optimized to minimize the negative log-likelihood of ground truth  $x_t$  during training.

### 2.2 Memory Augmentation

We consider memory as a set of context-target pairs  $\{(c_i, x_i)\}$  following Grave et al. (2017b); Khandelwal et al. (2020). These context-target pairs can be aggregated to obtain the next-token probability weighted by the similarity between hidden representations.<sup>4</sup> We formalize three types of context-target memories as follows:

**Local memory** The local memory is simply the preceding tokens in the same input. Specifically, for  $c_t = x_1, \dots, x_{t-1}$ , it is defined as:

$$\mathcal{M}_{\text{local}}(c_t) = \{(c_j, x_j)\}_{1 \leq j \leq t-1}. \quad (2)$$

Grave et al. (2017b) use the local memory at testing time, denoted by the “continuous cache” model. However, it has been argued less effective for Transformer-based models because they can already learn to leverage recent tokens in the self-attention layers (Khandelwal et al., 2020). Interestingly, we show that using local memory is still beneficial if we consider it during training.

**Long-term memory** Long-term memory denotes long-range context from the same document, but they cannot be directly accessed by attention. For example, if a document contains 10K tokens, only a short segment of text (e.g., 100-3K tokens) can be fed into a Transformer model because the complexity scales quadratically with the input

length. Formally, we divide a document into consecutive segments  $s^{(1)}, \dots, s^{(T)}$ , where a segment  $s^{(i)}$  contains  $L$  contexts  $s^{(i)} = \{c_1^{(i)}, \dots, c_L^{(i)}\}$ . The long-term memory for  $c_t^{(i)}$  is:

$$\mathcal{M}_{\text{long}}(c_t^{(i)}) = \{(c_j^{(k)}, x_j^{(k)})\}_{1 \leq k < i, 1 \leq j \leq L}. \quad (3)$$

Previous works (Dai et al., 2019; Rae et al., 2020; Martins et al., 2022; Ji et al., 2022; Wu et al., 2022; Lei, 2021) leverage hidden representations from previous segments with modified Transformer architectures to learn long-range dependency. Our approach does not modify the model architecture and is compatible with these neural architectures.<sup>5</sup>

**External memory** Finally, external memory assumes a large corpus  $\mathcal{D}$  and the external memory set can be defined as:

$$\mathcal{M}_{\text{ext}} = \{(c_j, x_j) \in \mathcal{D}\}. \quad (4)$$

$\mathcal{D}$  can be simply the training corpus, or a domain-specific corpus when the testing domain shifts (§5.3). Note that  $|\mathcal{M}_{\text{ext}}|$  is usually several orders of magnitude larger than previous two types (e.g.,  $10^8$ ); accessing all the memories is computationally expensive and requires approximate nearest neighbor search (Johnson et al., 2019).

## 3 Training with In-batch Memories

In this section, we propose a new training approach TRIME for language model training. Compared to standard language model training, our training objective assumes a set of *training memories*  $\mathcal{M}_{\text{train}} = \{(c_j, x_j)\}$ . We differentiate training memories from *testing memories*, as they are constructed on the fly during training and may deviate from the testing memories used during inference. Importantly, the training memories are constructed from the *same* training batch, which enables back-propagating the training signal to the current hidden representation as well as all the memory representations. We will discuss how to construct training memories in the next section (§4) and only discuss the training objective in a general form.

Our training objective is illustrated in Figure 1. Given a memory set  $\mathcal{M}$  and a context  $c$ , TRIME

<sup>4</sup>Other memory-augmented models differ in when the memory was introduced, such as using them in attention, and retrieve texts of different granularity as memory (Guu et al., 2020; Borgeaud et al., 2021).

<sup>5</sup>Note that continuous cache can be naturally extended to long-term memory, as we will experiment later. The earlier continuous cache work was applied to LSTMs on long sequences, as LSTMs can linearly scale with long sequences and there is no need to segment documents.

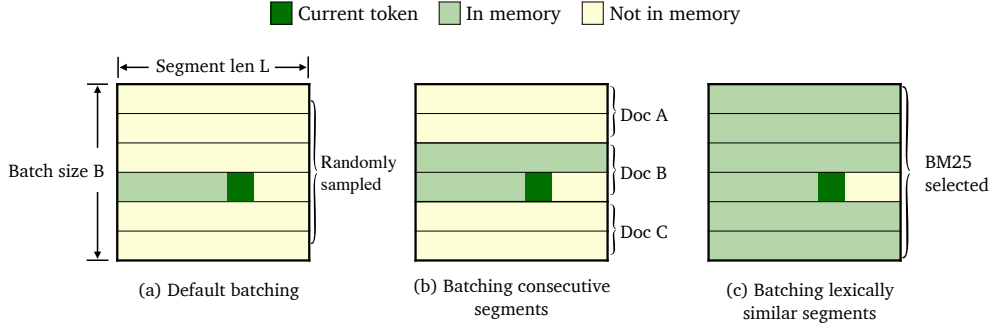


Figure 2: We present several data batching methods and memory construction strategies, in order to adapt to different sets of testing memories. (a) default batching: all the segments are randomly drawn from the training corpus (§4.1); (b) we batch consecutive segments from the same document ( $m > 1$ ) in one training batch to better leverage long-range contexts (§4.2); (c) we batch lexically-similar segments in one training batch selected by BM25 to better incorporate a large datastore at testing time (§4.3).

defines the next-token probability distribution as:

$$P(w | c) \propto \exp(E_w^\top f_\theta(c)) + \sum_{(c_j, x_j) \in \mathcal{M}_{\text{train}}: x_j = w} \exp(\text{sim}(g_\theta(c), g_\theta(c_j))). \quad (5)$$

Here,  $f_\theta(c)$  is the output representation of a Transformer model and  $E_w$  is the token embedding.  $g_\theta(\cdot)$  denotes the representations that can be used to compute similarity between  $c$  and all the contexts  $c_j$  in the memory  $\mathcal{M}_{\text{train}}$ . It is possible to simply take  $g_\theta = f_\theta$ ; however, we find that taking  $g_\theta$  to be the input of the final feed-forward layer in Transformer works better, which is consistent with the observation in Khandelwal et al. (2020). In addition,  $\text{sim}(\cdot, \cdot)$  is a similarity function and we found using the scaled dot-product  $\text{sim}(q, k) = \frac{q \cdot k}{\sqrt{d}}$  (Vaswani et al., 2017) leads to stable training and better performance in our preliminary experiments.

This training objective can be viewed as a contrastive loss (Hadsell et al., 2006): for a context-target pair  $(c, w^*)$ , the goal is to align the query representation  $f_\theta(c)$  (and  $g_\theta(c)$ ) with the *static* token representation  $E_{w^*}$ , and *contextualized* representations that share the same next token i.e.,  $g_\theta(c_j)$  for  $x_j = w^*$ . Our objective handles rare words nicely—if  $w^*$  does not appear in the training memory, the objective will fall back to aligning  $f_\theta(c)$  with only the word embedding  $E_{w^*}$ . Similar to the vanilla training loss (Eq. 1), our TRIME loss is optimized to minimize the negative log-likelihood of next token  $w^*$  and all the parameters  $\theta$  and  $E_w$  are updated during training.

Our training objective is also inspired by the success of contrastive learning in dense retrieval (Karpukhin et al., 2020). As we will show in §6, it can help improve retrieving contexts that share the same next token effectively when the

set of testing memories is large. Our objective is also closely connected to the objective used in Grave et al. (2017b); Khandelwal et al. (2020), which linearly interpolates the distribution of standard language modeling, and a distribution defined by cache/external datastore, e.g.,  $P(w | c) = (1 - \lambda)P_{\text{lm}}(w | c) + \lambda P_{\text{kNN}}(w | c)$ . Our work differs from previous works that we use this objective during *training* (and testing), while they only used it *at testing time*—the key is how to construct training memories that we will elaborate next.<sup>6</sup>

## 4 Adaption to Different Memories

**Inference** We are interested in incorporating the three types of memories defined in §2.2 and their combinations at testing time. The testing objective is basically the same as the training objective (Eq. 5) except that we take testing memories as a combination of  $\mathcal{M}_{\text{local}}$ ,  $\mathcal{M}_{\text{long}}$  and  $\mathcal{M}_{\text{ext}}$ . As  $\mathcal{M}_{\text{ext}}$  can be very large, we approximate it by retrieving the top-K closest terms to  $g_\theta(c)$ . We tune a temperature term  $\tau$  to adjust the weight of the memory component (see Appendix A for details).

**Notation** Throughout this section, we use  $L$  to denote segment length,  $B$  to denote the total number of segments used in the one training batch, and  $m$  to denote the number of consecutive segments from each document in the batch. Correspondingly, each batch will contain  $b \approx \frac{B}{m}$  different documents.  $L$ ,  $B$  and  $m$  are hyper-parameters that

<sup>6</sup>Grave et al. (2017b) described a “global normalization” variant in the paper, which is similar to our objective. However, they only used it at testing time and only considered short-term contexts in calculating the distribution. Other works (Merity et al., 2017; See et al., 2017) trained a pointer network with a learned gating component for the interpolation—we attempted training with a similar objective earlier and found it to perform worse than our current objective.



Model	#Params	Dev ( $\downarrow$ )	Test ( $\downarrow$ )	Speed ( $\uparrow$ )
Transformer (Baevski and Auli, 2019)	247M	17.96	18.65	-
+ continuous cache (Grave et al., 2017b)	247M	17.67	18.27	-
Transformer-XL (Dai et al., 2019)	257M	-	18.30	-
Transformer (our run)	247M	18.04	18.70	3.6k t/s
+ continuous cache	247M	17.65	18.26 $\pm 0.44$	3.6k t/s
★ TRIMELM	247M	17.10	17.76 $\pm 0.94$	3.6k t/s
★ TRIMELM <sub>long</sub>	247M	<b>17.01</b>	<b>17.64</b> $\pm 1.06$	3.6k t/s
kNN-LM (our run)	247M	16.40	16.37	300 t/s
+ continuous cache	247M	16.23	16.23 $\pm 0.14$	300 t/s
★ TRIMELM <sub>ext</sub> (w/o $\mathcal{M}_{\text{long}}$ )	247M	15.62	15.55 $\pm 0.82$	300 t/s
★ TRIMELM <sub>ext</sub>	247M	<b>15.51</b>	<b>15.41</b> $\pm 0.94$	300 t/s
kNN-LM (Khandelwal et al., 2020) <sup>†</sup>	247M	16.06	16.12	50 t/s
+ continuous cache (Grave et al., 2017b) <sup>†</sup>	247M	15.81	15.79 $\pm 0.33$	50 t/s
★ TRIMELM <sub>ext</sub> <sup>†</sup>	247M	<b>15.40</b>	<b>15.37</b> $\pm 0.75$	50 t/s

Table 2: Performance of our TRIME models on WIKITEXT-103 (247M models,  $L = 3,072$ ). <sup>†</sup>: the results are based on computing actual distances instead of using approximated distances returned by FAISS indexes, which requires a large SSD storage. To measure the speed of models (tokens/second), we run the model with a single NVIDIA RTX 3090 GPU and run the FAISS indexer with 32 CPUs.

we will choose for training, and will vary as we consider different memories during inference.

A key challenge is that the testing memories can be very large (e.g.,  $|\mathcal{M}_{\text{long}}| \sim 10^4$  and  $|\mathcal{M}_{\text{ext}}| \sim 10^8$  in our experiments) and it is computationally infeasible to keep training memories the same as testing memories. In the following, we will discuss three ways of constructing training memories and data batching, aiming to reduce the discrepancy between training and testing. Along the way, we will also present three major model instantiations: TRIMELM, TRIMELM<sub>long</sub>, TRIMELM<sub>ext</sub> (Table 1), which combine the training strategies and different sets of testing memories.

#### 4.1 Local Memory

$\mathcal{M}_{\text{local}}$  only considers all the previous tokens in the same segment. It is straightforward that we can simply use  $\mathcal{M}_{\text{train}} = \mathcal{M}_{\text{local}}$ . As shown in Fig. 2(a), we basically do not need to make *any* modifications compared to standard language model training. All we need is to replace the training objective of Eq. 1 by our objective in Eq. 5, by incorporating  $(c_j, x_j)$ ,  $\forall j < t$  in the memory during both training and testing. The computational overhead is also negligible compared to running neural encoders on the segment  $x_1, \dots, x_L$  itself. We denote this model as TRIMELM, which can be viewed as a lightweight replacement for vanilla language models. As we will show in the experiments, simply incorporating local memory provides a notable gain on multi-

ple LM benchmarks, showing the effectiveness of training with memories explicitly.

#### 4.2 Long-term Memory

In order to enable long-term memory augmentation, we pack multiple consecutive segments from the same document in a training batch (i.e.,  $m > 1$ ). For a context-target pair  $(c, w)$  in the training batch, its accessible memory  $\mathcal{M}_{\text{train}}$  includes tokens from previous segments as well as the preceding tokens in the same segment. Figure 2(b) illustrates the training batch construction and the training memory for a given token. At testing time, we can use a much longer context: we simply enumerate the number of segments used in  $\mathcal{M}_{\text{eval}}$  and choose the optimum based on the development set.

We denote this model as TRIMELM<sub>long</sub>. It shares a similar motivation with many previous works which aim to leverage memory from previous segments through attention recurrence (Dai et al., 2019; Ji et al., 2022), or memory compression (Rae et al., 2020; Martins et al., 2022; Wu et al., 2022). However, our solution deviates significantly from previous approaches. First, previous works need to store the hidden representations (of every layer) from previous segments and modify the self-attention layers to incorporate them. Our approach does not modify the architecture and only uses the outputs from the last layer. Additionally, previous works use stale memory representations and do not back-propagate gradients to the rep-

representations of previous segments, whereas our batching method enables gradient propagation to the memory and previous segments.<sup>7</sup> As we will show in the experiments, our approach is competitive with previous works while being conceptually simpler and computationally cheaper.

### 4.3 External Memory

Finally, we consider external memory  $\mathcal{M}_{\text{ext}}$ . Since  $\mathcal{M}_{\text{ext}}$  contains the context-target pairs in a large corpus such as the entire training set, we need to retrieve top- $K$  pairs from  $\mathcal{M}_{\text{ext}}$  measured by  $\text{sim}(g_\theta(c), g_\theta(c_j))$  through (approximate) similarity search (more details are given in §5.2).

Since the retrieved contexts at testing time are expected to be similar to the query context, we propose a simple heuristic for constructing training memories  $\mathcal{M}_{\text{train}}$  by packing segments that have large lexical overlap into the same batch using BM25 scores (Robertson and Zaragoza, 2009). Specifically, we start with a single segment and repeatedly add segments with highest BM25 scores into the same batch (Appendix B). A high BM25 score indicates that two segments have high lexical overlap and can serve as a good proxy to nearest neighbors in the external memory, which improves our model predictions at testing time.  $\mathcal{M}_{\text{train}}$  contains all tokens from other segments as well as the previous tokens in the same segment (Figure 2(c)). We set  $m = 1$  during training as many segments from the same document tend to have high lexical overlap and denote this model by  $\text{TRIMELM}_{\text{ext}}$ .

In practice, when considering tokens from both the current segment and other segments in the batch, we observe that the model tends to leverage local memory more and ignore other segments. To encourage the use of information from other segments, we exclude the local memory from  $\mathcal{M}_{\text{train}}$  with a probability of  $p$  during training (we find that  $p = 90\%$  works the best, see Appendix H). This significantly improves performance when the model is evaluated with a large set of external memory.

## 5 Experiments

### 5.1 Datasets and Tasks

We evaluate our approach on two popular language modeling benchmarks: WIKITEXT-103 (Merity

et al., 2017), ENWIK8 (Mahoney, 2009), and a machine translation benchmark: IWSLT’14 De-En. We also evaluate domain-adaptation performance on the BOOKSCORPUS dataset (Zhu et al., 2015).

**WIKITEXT-103** is a word-level language modeling dataset consisting of 103M training tokens. We evaluate on two model configurations: one uses a 247M Transformer model and a segment length  $L = 3,072$  and another one uses a 150M Transformer model with a segment length  $L = 150$ .

**ENWIK8** is a character-level language modeling dataset that contains a total of 100M characters. We use a 12-layer Transformer model with a hidden dimension 512 and segment length  $L = 512$ .

**BOOKSCORPUS** is a word-level language modeling dataset. We build our own train/dev/test splits which consist of 100M/250K/250K tokens. On this dataset, we evaluate the models trained on WIKITEXT-103 to study how our approach can adapt to new domain without re-training.

**IWSLT’14 De-En** is a machine translation task, which consists of 170K translation pairs. We use a Transformer encoder-decoder model. See Appendix C for how we adapt our approach to the machine translation task.

See Appendix C for data statistics and task setups and Appendix D for model configurations.

### 5.2 Training and Inference Details

We implement our approach using the Fairseq library (Ott et al., 2019). For  $\text{TRIMELM}_{\text{long}}$  and  $\text{TRIMELM}_{\text{ext}}$ , we tune the number of segments used in  $\mathcal{M}_{\text{long}}$  on the development set during evaluation. Our  $\text{TRIMELM}_{\text{ext}}$  model requires building a large datastore at testing time and we use the FAISS library (Johnson et al., 2019) for approximate nearest neighbor search (details in Appendix D).

We first train our model with the standard LM objective (Eq. 1) for the first 5% updates. Without this warmup stage, we observe the training process to be unstable probably due to a large variance in the estimated distributions. We use different memories when evaluating different instantiations of TRIME, as shown in Table 1. We find that when a large set of external memory  $\mathcal{M}_{\text{ext}}$  is considered during inference, the performance can be improved by linearly interpolating the output distribution and a distribution over the memory, similarly to kNN-LM (Khandelwal et al., 2020). Thus, we apply an additional linear interpolation to our output probability distribution when considering external mem-

<sup>7</sup>We also attempted using segments in previous training batches as stale representations and did not find any improvement in preliminary experiments.

Model	Dev ( $\downarrow$ )	Test ( $\downarrow$ )
Transformer	28.11	29.14
Transformer-XL	23.42	24.56
Compressive Transformer	-	24.41
$\infty$ -former	-	24.22
LaMemo	22.98	23.77
Transformer (our run)	25.31	25.87
+ continuous cache*	22.95	23.59 $\pm 2.28$
★ TRIMELM	24.45	25.60 $\pm 0.27$
★ TRIMELM <sub>long</sub>	<b>21.76</b>	<b>22.66</b> $\pm 3.21$

Table 3: Performance on the WIKITEXT-103 dataset (150M models,  $L = 150$ ). TRIMELM<sub>long</sub> uses a long-term memory with 15,000 tokens. Transformer-XL: (Dai et al., 2019), Compressive Transformer: (Rae et al., 2020),  $\infty$ -former: (Martins et al., 2022), LaMemo: (Ji et al., 2022). \*: cache adapted to long-term memory.

ory  $\mathcal{M}_{\text{ext}}$  (see Appendix A for details).

### 5.3 Results: Language Modeling

**TRIMELM vs. vanilla LM** We first compare our TRIMELM model which only uses local memory during training and testing. Table 2 shows that adding a continuous cache during inference can improve the performance of vanilla Transformer from 18.70 to 18.26, and our TRIMELM further improves the perplexity to 17.76. These results suggest that even though the attention mechanism can “see” local context, using local memory during both training and testing can still improve model performance. TRIMELM has no computational overhead compared to vanilla LM (indicated by the “speed” column), making it a simple and better replacement for vanilla language models. Similar trends can be observed in Table 3 and Table 4 (25.87 vs. 25.60 and 1.16 vs. 1.12). The improvement is much smaller though, due to a much smaller segment length  $L$ . More analysis is given in Appendix G.

**TRIMELM<sub>long</sub> leverages long contexts** We then examine our TRIMELM<sub>long</sub> model which is trained with the data batching method described in §4.2. As shown in Table 3 and Table 4, TRIMELM<sub>long</sub> improves vanilla Transformer models substantially (i.e., 25.87  $\rightarrow$  22.66 on WIKITEXT-103 and 1.16  $\rightarrow$  1.05 on ENWIK8) by leveraging long-range contexts at inference time. We find the model achieves its best results when leveraging 15,000 tokens on WIKITEXT-103 and 24,576 tokens on ENWIK8, even though the segments used during training are much shorter ( $L = 150$  and 512 respectively). We also add continuous

Model	#Params	Dev ( $\downarrow$ )	Test ( $\downarrow$ )
T12	44M	-	1.11
Transformer-XL	41M	-	1.06
Adapt-Span	39M	1.04	1.02
Longformer	41M	1.02	1.00
Expire-Span	38M	-	<b>0.99</b>
Transformer ( $L = 512$ )	38M	1.18	1.16
+ continuous cache*	38M	1.16	1.17 $\pm 0.01$
★ TRIMELM	38M	1.14	1.12 $\pm 0.04$
★ TRIMELM <sub>long</sub>	38M	<b>1.08</b>	<b>1.05</b> $\pm 0.11$
SRU++ ( $L = 512$ )	42M	1.05	1.03
★ TRIMELM <sub>long</sub>	42M	<b>1.03</b>	<b>1.01</b> $\pm 0.02$
SRU++ ( $L = 2,048$ )	42M	1.02	0.99
★ TRIMELM <sub>long</sub>	42M	<b>1.00</b>	<b>0.98</b> $\pm 0.01$

Table 4: Performance on the ENWIK8 dataset. TRIMELM<sub>long</sub> achieves the best results by using a long-term memory of a size 24,576. T12: (Al-Rfou et al., 2019), Transformer-XL: (Dai et al., 2019), Adapt-Span: (Sukhbaatar et al., 2019), Longformer: (Beltagy et al., 2020), Expire-Span: (Sukhbaatar et al., 2021), SRU++: (Lei, 2021). \*: cache adapted to long-term memory.

cache to the vanilla Transformer model and find it to underperform our model, demonstrating the importance of joint training using our approach.

Compared to previous methods which explicitly leverage hidden representations from previous segments (Dai et al., 2019; Rae et al., 2020; Martins et al., 2022; Ji et al., 2022; Lei, 2021), our approach achieves better or at least competitive performance. Different from these approaches which need to store all the hidden representations of every layer and modify the model architecture, we only incorporate the outputs from the last layer—requiring less computations and GPU memory. Our approach is orthogonal and can be applied on top of these models. To verify this, we adapt our approach to SRU++ (Lei, 2021) (see details in Appendix E). As shown in the bottom block of Table 4, TRIMELM<sub>long</sub> gains consistently improvement over vanilla SRU++, outperforming previously reported results given the same model size.

**TRIMELM<sub>ext</sub> vs. kNN-LM** Finally, our model TRIMELM<sub>ext</sub> outperforms the kNN-LM model (Khandelwal et al., 2020), which uses external memory only at testing time—improving the perplexity from 16.23 to 15.41 on WIKITEXT-103 (Table 2). We also evaluate a model which does not use long-term memory (denoted by TRIMELM<sub>ext</sub> w/o  $\mathcal{M}_{\text{long}}$ ) for a fair comparison with kNN-LM with continuous cache and the difference is very small (15.55 vs 15.41). Our results suggest that by using contrastive loss and BM25 batching (§4.3),

Model	$\mathcal{M}_{\text{ext}}$	Dev ( $\downarrow$ )	Test ( $\downarrow$ )
Transformer	-	62.72	53.98
★ TRIMELM	-	59.39	49.25
★ TRIMELM <sub>long</sub>	-	49.21	<b>39.50</b>
kNN-LM + cont. cache	WIKI	53.27	43.24
★ TRIMELM <sub>ext</sub>	WIKI	47.00	37.70
kNN-LM + cont. cache	BOOKS	42.12	32.87
★ TRIMELM <sub>ext</sub>	BOOKS	36.97	<b>27.84</b>

Table 5: Domain-adaptation performance on the BOOKSCORPUS dataset. All models are trained on WIKITEXT-103 and evaluated on BOOKSCORPUS without re-training or fine-tuning and we consider using WIKITEXT-103 and BOOKSCORPUS to build the external datastore respectively. We use a long-term memory of a size 49,152 for TRIMELM<sub>long</sub>, TRIMELM<sub>ext</sub>, and continuous cache in this experiment.

the model learns to better retrieve and leverage information from a large external memory.

**Domain adaptation** We evaluate the domain-adaptation performance of TRIME on BOOKSCORPUS (Zhu et al., 2015). We take models that are trained on WIKITEXT-103 and evaluate them on BOOKSCORPUS without any re-training or fine-tuning. As shown in Table 5, a vanilla Transformer model trained on WIKITEXT-103 performs poorly on BOOKSCORPUS. TRIMELM and TRIMELM<sub>long</sub> can significantly improve the performance as they leverage local or long-term memory to adapt to the new domain. By building the external memory using BOOKSCORPUS, both kNN-LM and TRIMELM<sub>ext</sub> perform much better on BOOKSCORPUS compared to the vanilla Transformer model. TRIMELM<sub>ext</sub> outperforms kNN-LM on domain adaptation. This indicates that although the memory representations are optimized on one domain, our approach does not overfit, and building an external memory using the target domain dataset enables the model to perform well with domain shifts.

#### 5.4 Results: Machine Translation

To showcase the generality of our training approach TRIME to other generation tasks, we evaluate our approach on the IWSLT’14 de-en translation task. Since it is a sentence-level task, we do not use any local or long-term memory ( $\mathcal{M}_{\text{local}}$ ,  $\mathcal{M}_{\text{long}}$ ), as there are few repetitive tokens. We denote our model as TRIMEMT<sub>ext</sub>.

As shown in Table 6, our approach improves the vanilla Transformer by 1.15 BLEU score and outperforms kNN-MT (Khandelwal et al., 2021). This

Model	BLEU ( $\uparrow$ )
Transformer enc-dec	32.58
kNN-MT	33.15 $\uparrow$ 0.57
★ TRIMEMT <sub>ext</sub>	<b>33.73</b> $\uparrow$ 1.15

Table 6: Results on the IWSLT’14 De-En test set. We adapt TRIME to the machine translation task. We use a beam size of 4 during evaluation.

Method	Test memory		
	$\mathcal{M}_{\text{local}}$	$\mathcal{M}_{\text{local}}, \mathcal{M}_{\text{long}}$	$\mathcal{M}_{\text{local}}, \mathcal{M}_{\text{long}}, \mathcal{M}_{\text{ext}}$
TRIMELM	<b>17.10</b>	17.17	17.40
TRIMELM <sub>long</sub>	17.12	<b>17.01</b>	16.48
TRIMELM <sub>ext</sub>	17.99	17.80	<b>15.51</b>

Table 7: Evaluating our three models (w/ different training methods) on different sets of testing memories. The results are based on the development set of WIKITEXT-103 (247M models,  $L = 3,072$ ).

demonstrates that our approach is able to improve the performance on other language generation tasks with different memory access.

## 6 Analysis

We conduct ablation studies and analysis to further understand individual components of our approach. Due to the limited computation budget, some experiments on WIKITEXT-103 are conducted with a small 7M Transformer model (8 layers, hidden dimension 128) in this section and the trends are generally similar for smaller models (see Appendix D and Appendix F for details).

**Memory construction** We first study how different data batching and memory construction strategies affect the performance when different testing memories are used. We compare our three models (TRIMELM, TRIMELM<sub>long</sub>, TRIMELM<sub>ext</sub>) in Table 7. This ablation study clearly shows that packing consecutive segments and segments with high BM25 scores in the same training batch and constructing memories properly can improve the performance when the long-range and external memories are used. This demonstrates the importance of closing the gap between training and inference.

**Leveraging long-range contexts** We study if our model is able to handle large long-term memory. As Figure 3 shows, our model is able to effectively handle long-range context (more than 10k tokens), which goes beyond typical attention context. Compared to continuous cache (Grave et al., 2017b,a), the improvement of our approach be-



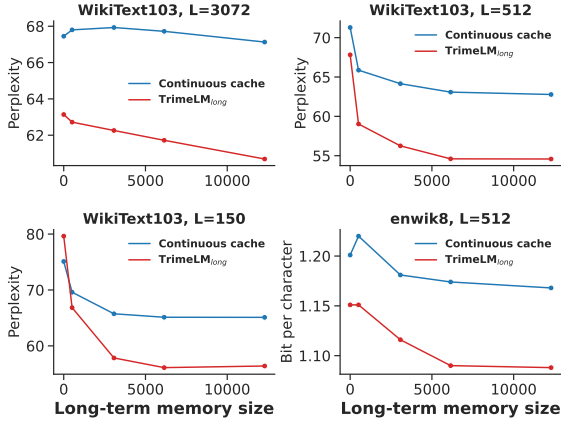


Figure 3: Performance of TRIMELM<sub>long</sub> (with different segment lengths  $L$ ) and continuous cache (adapted to long-term memory) on the WIKITEXT-103 (7M models) and ENWIK8 development sets.

comes larger when more long-term memory is incorporated. This suggests that our model is able to leverage long-range context much more effectively.

**Additional analysis** We conduct more ablation studies and analysis in Appendix G. We summarize them as follows. (1) Our ablation studies show using BM25 batching method and enabling back-propagation to update memory representations are important for our approach (Table 11). (2) TRIMELM is able to leverage local memory effectively to improve performance with different segment lengths  $L$  (Table 12). (3) TRIMELM<sub>ext</sub> outperforms kNN-LM in terms of top-K retrieval accuracy given the external memory set (Table 13). (4) We study the perplexity of tokens in different frequency groups and find that TRIMELM and TRIMELM<sub>long</sub> achieve larger improvements on rare words while TRIMELM<sub>ext</sub> improves results across the board (Table 14).

## 7 Related Work

**Memory-augmented language models** We have discussed continuous cache, kNN-LM and models that leverage representations from long-range context in the previous sections. Yogatama et al. (2021) also aim to combine several types of memories by learning an adaptive gating function; however, their external memory uses a pre-trained vanilla language model. Borgeaud et al. (2021) demonstrate a remarkable performance by augmenting LMs with an external datastore of trillion of tokens and their datastore is built based on chunks of text using off-the-shelf BERT embeddings (Devlin et al., 2019). Our approach differs from prior works in the following aspects:

(1) we update the memory representations through back-propagation from the end loss; (2) our model does not modify the base architecture; (3) we consider different types of memories in a unified framework. GNN-LM (Meng et al., 2022) augments LMs with a graph neural network to aggregate information of retrieved items from external memory, which makes an orthogonal contribution to our paper.

**Transformers for long inputs** A large body of research has investigated how to scale self-attention mechanism to long contexts, either through sparse attention (Liu et al., 2018; Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020) or sub-quadratic-time attention (Wang et al., 2020; Choromanski et al., 2020; Peng et al., 2021; Katharopoulos et al., 2020). See Tay et al. (2020) for a comprehensive survey of efficient Transformers. Our approach is orthogonal, as we only change the training objective and data batching to enable models to use large contexts during inference.

**Memory-augmented models for downstream tasks** Prior works have also improved models for downstream tasks with a retrieval component, such as question answering (Kumar et al., 2016; de Masson D’Autume et al., 2019; Karpukhin et al., 2020; Guu et al., 2020; Zemlyanskiy et al., 2021; de Jong et al., 2022; Chen et al., 2022; Izacard and Grave, 2021; Singh et al., 2021), dialogue (Fan et al., 2021), and other knowledge-intensive NLP tasks (Lewis et al., 2020; Petroni et al., 2021). Notably, recent works (de Jong et al., 2022; Chen et al., 2022) explore a similar idea for question answering and leverage in-batch memories to train memory representations for entity mentions or QA pairs, which are further incorporated into Transformers at a second stage.

## 8 Conclusion

In this work, we propose TRIME, a training approach for language modeling. We present three model instantiations TRIMELM, TRIMELM<sub>long</sub>, TRIMELM<sub>ext</sub>: Through carefully-designed data batching and memory construction during training, we show that our models can leverage long-range contexts and external memory effectively at testing time. Our approach adds little computational overhead and does not modify model architectures, making it compatible with other neural models and techniques. For future work, we are interested in training TRIME with large language models and other text generation tasks.

## Limitations

We discuss limitations of our research as follows.

- Despite the strong performance achieved by our approach when incorporating a large set of external memory, it results in a reduced inference efficiency at the same time due to the nearest neighbor search. For example, the model is 10× slower when incorporating external memory. This issue can be more crucial when the external memory is even larger. Potential solutions to this issue include (1) constructing the memory using a coarser granularity (e.g., text blocks) (Borgeaud et al., 2021); (2) compressing the external memory set and reducing the dimension of memory representations (He et al., 2021).
- We mainly experiment with Transformer-based models and additionally adapt our approach to SRU++ (Lei, 2021). We believe our approach is compatible with other architectures or techniques such as Transformer-XL (Dai et al., 2019) and Compressive Transformer (Rae et al., 2020). We plan to explore them as future work.
- We evaluate our approach on machine translation to test the generality of TRIME to other generation tasks. However, due to compute limitation, we only evaluate it on a small dataset (i.e., IWSLT’14), which consists of 4M tokens in the external memory. We leave the evaluation on larger machine translation datasets as future work.
- Our paper mainly studies language modeling tasks and machine translation tasks. Although we believe our approach is compatible with all language generation tasks, how to adapt TRIME to natural language understanding tasks such as text classification still remains an open question.
- The biggest model we experimented with consists of 247M parameters due to our compute limit. The state-of-the-art auto-regressive LMs contain hundreds of billions of parameters (Brown et al., 2020). We hope to see future efforts in scaling up our approach and evaluating the effectiveness on large LMs.

## Ethical Considerations

Our proposed approach leverages external memory to achieve strong results on multiple language modeling benchmarks. In our experiments, we construct the external memory using the corpus on which the model is trained, while it can be constructed using any corpus. In general, we suggest practitioners constructing external memory using a public corpus, as retrieving from the external datastore can cause information leakage from the corpus. We acknowledge this ethical consideration and caution those who apply our approach to privacy-sensitive domains.

## Acknowledgments

We thank Jane Pan, Howard Chen, Alexander Wetzig, Tianyu Gao, Kaiyu Yang, Mengzhou Xia, Jinyuk Lee, and the members of Princeton NLP group for helping with proofreading and providing valuable feedback. This research is partially supported by the James M. \*91 Research Innovation Fund for Data Science and a gift from Apple. ZZ is also supported by a JP Morgan PhD fellowship.

## References

- Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2019. Character-level language modeling with deeper self-attention. In *Conference on Artificial Intelligence (AAAI)*, volume 33, pages 3159–3166.
- Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *International Conference on Learning Representations (ICLR)*.
- Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document Transformer. *arXiv preprint arXiv:2004.05150*.
- Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2021. Improving language models by retrieving from trillions of tokens. *arXiv preprint arXiv:2112.04426*.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1877–1901.
- Wenhu Chen, Pat Verga, Michiel de Jong, John Wieting, and William Cohen. 2022. Augmenting

- pre-trained language models with qa-memory for open-domain question answering. *arXiv preprint arXiv:2204.04581*.
- Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse Transformers. *arXiv preprint arXiv:1904.10509*.
- Krzysztof Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with Performers. *arXiv preprint arXiv:2009.14794*.
- Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.
- Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Association for Computational Linguistics (ACL)*.
- Michiel de Jong, Yury Zemlyanskiy, Nicholas FitzGerald, Fei Sha, and William Cohen. 2022. Mention memory: incorporating textual knowledge into transformers through entity mention attention. In *International Conference on Learning Representations (ICLR)*.
- Cyprien de Masson D’Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. 2019. Episodic memory in lifelong language learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 32.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional Transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.
- Angela Fan, Claire Gardent, Chloé Braud, and Antoine Bordes. 2021. Augmenting Transformers with knn-based composite memory for dialog. *Transactions of the Association of Computational Linguistics (TACL)*, 9:82–99.
- Edouard Grave, Moustapha M Cisse, and Armand Joulin. 2017a. Unbounded cache model for online language modeling with open vocabulary. *Advances in Neural Information Processing Systems (NIPS)*, 30.
- Edouard Grave, Armand Joulin, and Nicolas Usunier. 2017b. Improving neural language models with a continuous cache. In *International Conference on Learning Representations (ICLR)*.
- Kelvin Guu, Kenton Lee, Zora Tung, Panupong Papat, and Ming-Wei Chang. 2020. REALM: Retrieval-augmented language model pre-training. In *International Conference on Machine Learning (ICML)*.
- Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 1735–1742.
- Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. 2021. Efficient nearest neighbor language models. In *Empirical Methods in Natural Language Processing (EMNLP)*.
- Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *European Chapter of the Association for Computational Linguistics (EACL)*.
- Haozhe Ji, Rongsheng Zhang, Zhenyu Yang, Zhipeng Hu, and Minlie Huang. 2022. LaMemo: Language modeling with look-ahead memory. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.
- Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.
- Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Empirical Methods in Natural Language Processing (EMNLP)*.
- Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are RNNs: Fast autoregressive Transformers with linear attention. In *International Conference on Machine Learning (ICML)*, pages 5156–5165.
- Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2021. Nearest neighbor machine translation. In *International Conference on Learning Representations (ICLR)*.
- Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations (ICLR)*.
- Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning (ICML)*, pages 1378–1387.
- Tao Lei. 2021. When attention meets fast recurrence: Training language models with reduced compute. In *Empirical Methods in Natural Language Processing (EMNLP)*.

- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:9459–9474.
- Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating Wikipedia by summarizing long sequences. In *International Conference on Learning Representations (ICLR)*.
- Matt Mahoney. 2009. Large text compression benchmark.
- Pedro Henrique Martins, Zita Marinho, and André FT Martins. 2022.  $\infty$ -former: Infinite memory Transformer. In *Association for Computational Linguistics (ACL)*.
- Yuxian Meng, Shi Zong, Xiaoya Li, Xiaofei Sun, Tianwei Zhang, Fei Wu, and Jiwei Li. 2022. Gnn-lm: Language modeling based on global contexts via gnn. In *International Conference on Learning Representations (ICLR)*.
- Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)*.
- Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Association for Computational Linguistics (ACL)*, pages 311–318.
- Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. 2021. Random feature attention. In *International Conference on Learning Representations (ICLR)*.
- Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, et al. 2021. KILT: a benchmark for knowledge intensive language tasks. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 2523–2544.
- Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.
- Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2020. Compressive Transformers for long-range sequence modelling. In *International Conference on Learning Representations (ICLR)*.
- Stephen Robertson and Hugo Zaragoza. 2009. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc.
- Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Association for Computational Linguistics (ACL)*, pages 1073–1083.
- Devendra Singh, Siva Reddy, Will Hamilton, Chris Dyer, and Dani Yogatama. 2021. End-to-end training of multi-document reader and retriever for open-domain question answering. *Advances in Neural Information Processing Systems (NIPS)*.
- Sainbayar Sukhbaatar, Édouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in Transformers. In *Association for Computational Linguistics (ACL)*, pages 331–335.
- Sainbayar Sukhbaatar, Da Ju, Spencer Poff, Stephen Roller, Arthur Szlam, Jason Weston, and Angela Fan. 2021. Not all memories are created equal: Learning to forget by expiring. In *International Conference on Machine Learning (ICML)*, pages 9902–9912. PMLR.
- Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient Transformers: A survey. *arXiv preprint arXiv:2009.06732*.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems (NIPS)*, 30.
- Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.
- Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. 2022. Memorizing Transformers. In *International Conference on Learning Representations (ICLR)*.
- Dani Yogatama, Cyprien de Masson d’Autume, and Lingpeng Kong. 2021. Adaptive semiparametric language models. *Transactions of the Association of Computational Linguistics (TACL)*, 9:362–373.
- Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big Bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 17283–17297.
- Yury Zemlyanskiy, Joshua Ainslie, Michiel de Jong, Philip Pham, Ilya Eckstein, and Fei Sha. 2021. Readtwice: Reading very large documents with



memories. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. OPT: Open pre-trained Transformer language models. *arXiv preprint arXiv:2205.01068*.

Xin Zheng, Zhirui Zhang, Junliang Guo, Shujian Huang, Boxing Chen, Weihua Luo, and Jiajun Chen. 2021. Adaptive nearest neighbor machine translation. In *Association for Computational Linguistics (ACL)*, pages 368–374.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *International Conference on Computer Vision (ICCV)*, pages 19–27.

## A Inference Method

**Testing objective** Formally speaking, our testing objective is basically the same as the training objective (Eq. 5):

$$P(w | c) \propto \exp(E_w^\top f_\theta(c)) + \sum_{(c_j, x_j) \in \mathcal{M}_{\text{eval}}: x_j = w} \exp\left(\frac{\text{sim}(g_\theta(c), g_\theta(c_j))}{\tau}\right), \quad (6)$$

except that we take  $\mathcal{M}_{\text{eval}}$  as a combination of  $\mathcal{M}_{\text{local}}$ ,  $\mathcal{M}_{\text{long}}$  and  $\mathcal{M}_{\text{ext}}$ . As  $\mathcal{M}_{\text{ext}}$  can be very large, we approximate it by retrieving the top-K closest terms to  $g_\theta(c)$ . Formally,  $\mathcal{M}_{\text{eval}}$  of three instantiations of TRIME is constructed as follows,

$$\mathcal{M}_{\text{eval}} = \begin{cases} \mathcal{M}_{\text{local}} & (\text{TRIMELM}) \\ \mathcal{M}_{\text{local}} \cup \mathcal{M}_{\text{long}} & (\text{TRIMELM}_{\text{long}}) \\ \mathcal{M}_{\text{local}} \cup \mathcal{M}_{\text{long}} \cup \text{kNN}(\mathcal{M}_{\text{ext}}, g_\theta(c)) & (\text{TRIMELM}_{\text{ext}}) \end{cases} \quad (7)$$

where  $\text{kNN}(\mathcal{M}_{\text{ext}}, g_\theta(c))$  returns the top-K closest terms to  $g_\theta(c)$  in the memory set  $\mathcal{M}_{\text{ext}}$ . Additionally, because  $\mathcal{M}_{\text{eval}}$  may be different from the training memories, we tune a temperature term  $\tau$  to adjust the weight of the memory component when calibrating the distribution, based on the development set.

**Linear interpolation when using  $\mathcal{M}_{\text{ext}}$**  We find that when a large set of external memory  $\mathcal{M}_{\text{ext}}$  is considered during inference, the performance can be improved by calibrating a separated distribution over the memory and interpolating the output distribution and the memory distribution, similarly to kNN-LM (Khandelwal et al., 2020). We think this is because the distribution of the similarity values has been significantly shifted during inference, while the relative ranking preserves. As a result, having values from two different distributions in one softmax normalization is sub-optimal compared to computing two separated probabilities and interpolating them.

Thus, we apply an additional linear interpolation to our output probability distribution. Specifically, we first use Eq. 6 to compute the distribution  $P(w | c)$ . Then, we compute a probability distribution over the tokens in memory  $P'(w | c)$  as follow,

$$P'(w | c) \propto \sum_{(c_j, x_j) \in \mathcal{M}_{\text{eval}}: x_j = w} \exp\left(\frac{\text{sim}(g_\theta(c), g_\theta(c_j))}{\tau'}\right). \quad (8)$$

We linearly interpolate these two probability distributions with a coefficient  $\lambda$  and get the final output

$$P_{\text{final}}(w | c):$$

$$P_{\text{final}}(w | c) = (1 - \lambda)P(w | c) + \lambda P'(w | c). \quad (9)$$

We tune the temperature terms and  $\lambda$  on the development set.

---

**Algorithm 1:** Packing segments using BM25 scores.  $\text{SimSeg}(I, c, k)$  returns the top- $k$  most similar segments to  $c$  in the BM25 indexer  $I$ . ( $k = 20$  when packing segments in our experiments.)

---

**Data:** training segments  $S = \{s_1, \dots, s_{|S|}\}$

**BM25 Indexer:**  $I$

**Hyper-parameters:**  $k$ , batch size  $B$

**Output:** training batches  $T$

```

 $l \leftarrow \text{list}();$ 
 $c \leftarrow \text{None};$ 
while  $|S| \neq 0$  do
  if  $c$  is None then
     $c \leftarrow \text{random\_sample}(S);$ 
  end
   $l.\text{append}(c);$ 
   $S.\text{remove}(c);$ 
   $n \leftarrow \text{None};$ 
  for  $c'$  in  $\text{SimSeg}(I, c, k)$  do
    if  $c'$  in  $S$  then
       $n \leftarrow c';$ 
      break;
    end
  end
   $c \leftarrow n;$ 
end
 $T \leftarrow \{[l_1, \dots, l_B], [l_{B+1}, \dots, l_{2B}], \dots\};$ 
return  $T;$ 

```

---

## B Packing Segments Using BM25 Scores

In §4.3, we construct training memories  $\mathcal{M}_{\text{train}}$  by packing segments that have large lexical overlap into the same batch using BM25 (Robertson and Zaragoza, 2009). Algorithm 1 shows the process to pack segments into training batches. We start with a single segment and repeatedly add segments with highest BM25 scores into the same batch.

## C Dataset Statistics and Tasks

We evaluate our approach on three benchmarks: WIKITEXT-103, ENWIK8, and IWSLT'14. We also evaluate our approach on BOOKSCORPUS for

	Train	Dev	Test	$ \mathcal{V} $	len
WIKITEXT-103	110M	0.2M	0.3M	270K	3.6K
ENWIK8	94M	5.2M	5.2M	256	-
BOOKSCORPUS	100M	250K	250K	270K	90K
IWSLT'14	160K	7K	6K	16K	25

Table 8: Statistics of the four datasets used in our paper. WIKITEXT-103 and BOOKSCORPUS are a word-level LM task and ENWIK8 is a character-level language modeling task, and IWSLT'14 is a German-English machine translation task. len: denotes the average number of tokens over training examples in each dataset. When evaluating models on BOOKSCORPUS without re-training, we use the WIKITEXT-103's vocabulary. IWSLT'14 is a sentence-level task, so incorporating long-range context will not help.

domain adaptation (Appendix 5.3). Table 8 shows the statistics.

**WIKITEXT-103** (Merity et al., 2017) is a word-level language modeling dataset consisting of 103M training tokens. Following standard practice, we use adaptive softmax and adaptive token embeddings (Baevski and Auli, 2019) in our model and report perplexity. In order to better compare with previous work, we evaluate on two model configurations—one uses a 247M Transformer model and a segment length  $L = 3,072$  following Baevski and Auli (2019); Khandelwal et al. (2020) and another one uses a 150M Transformer model with segment length  $L = 150$  following Dai et al. (2019). More details are provided in Appendix D.

**ENWIK8** (Mahoney, 2009) is a character-level language modeling dataset that contains a total of 100M characters. Following previous work, we report bit-per-character (bpc) on this dataset. We use a 12-layer Transformer model with a hidden dimension 512 and segment length  $L = 512$ .

We also evaluate the **IWSLT'14** DE→EN machine translation task, which consists of 170K translation pairs. Following Khandelwal et al. (2021), we build an external memory by taking all the translation contexts and the corresponding target token  $((x, y_{<t}), y_t)$  on the training set. We use the output representation as  $f((x, y_{<t}))$  and the input representation of last FFN layer as  $g((x, y_{<t}))$  to compute the loss. Similarly, we use BM25 to batch training data – we encourage two target sentences with a high BM25 score to be in the same training batch (see Algorithm 1). We use the default model configuration in the Fairseq library (Ott

et al., 2019), and sacrebleu (Post, 2018) to compute BLEU scores (Papineni et al., 2002).

We evaluate our approach for domain adaptation on the **BOOKSCORPUS** dataset (Zhu et al., 2015), which is a word-level language modeling dataset. The complete BOOKSCORPUS dataset consists of 0.7B tokens. We build our own train/dev/test splits which consist of 100M/250K/250K tokens respectively. The train set is only used to build external memory. On this dataset, we evaluate the models trained on WIKITEXT-103 to study how our approach can adapt to new domain without re-training or fine-tuning. The model we used on this dataset is the 247M Transformer model with a segment length  $L = 3,072$ .

## D Model Configurations and Hyperparameters

Table 9 shows the model configurations and hyperparameters that we used in our experiments. Following Baevski and Auli (2019), during training, we train the model with fixed-length segments; during evaluation, we evaluate on the tokens at the end of the segment (i.e., an evaluation segment can overlap with others).

When evaluating with large external memory, we always retrieve top- $K$  ( $K = 1,024$ ) context-target pairs for language modeling. For machine translation, we tune  $K = \{1, 2, 4, 8, 16, 32, 64\}$  following Zheng et al. (2021).

## E Applying TRIMELM<sub>long</sub> to SRU++

We apply our approach to SRU++ (Lei, 2021) and we believe our approach is also compatible with other architectures such as Transformer-XL (Dai et al., 2019). SRU++ is a language model which combines recurrent units and the attention mechanism. SRU++ use hidden representations from the previous segment at attention layers to incorporate long-range contexts, similarly to Dai et al. (2019).

To apply our approach to SRU++, we follow their data-batching method as it is required due to the recurrence of the model architecture. We construct the training memory using all the contexts in the current segment (i.e., local memory) and all contexts in the previous segment (i.e., long memory). Note that the memory representations from the previous segment will be stale, thus we do not back-propagate to that part. During training, we update the model with 400K steps and a batch size of

Dataset	WIKITEXT-103			ENWIK8	IWSLT'14
<b>Model</b>					
#Params	247M	150M	7M	38M	39M
#Layers	16	16	8	12	6+6
Hidden dimension	1024	410	128	512	512
FFN intermediate dimension	4096	2100	512	2048	1024
Adaptive softmax?	yes	yes	yes	no	no
<b>Training</b>					
Segment length	3072	150	3072	512	-
#Tokens per update	73728	36000	24576	49152	16384
Gradient accumulation	3	4	1	4	1
Batch size per update	24	240	8	96	-
#Consecutive segments	4	60	8	24	-
#In-batch memories	24576	9000	24576	12288	-
<b>Evaluation</b>					
Segment length	512	64	512	80	-
#Optimal long-term memories	12288	15000	12288	24576	-
<b>Optimizer and scheduler</b>					
Optimizer type	nag	adam	adam	adam	adam
Learning rate	1.0	5e-4	5e-4	2.5e-4	5e-4
Grad crop norm	0.1	0.0	0.0	0.0	0.0
Update steps	286000	200000	200000	400000	170000
Scheduler type	cosine	inverse_sqrt	inverse_sqrt	cosine	cosine
Linear warmup steps	16000	8000	8000	0	4000

Table 9: Model configurations and hyperparameters in our experiments.

Model	Dev ( $\downarrow$ )	Test ( $\downarrow$ )
Transformer	82.68	83.66
+ continuous cache	67.45	68.08
kNN-LM	51.88	52.24
+ continuous cache	45.15	45.82
★ TRIMELM	54.78	54.69 $\downarrow$ 28.97
★ TRIMELM <sub>long</sub>	60.71	60.10 $\downarrow$ 23.56
★ TRIMELM <sub>ext</sub>	<b>41.50</b>	<b>42.36</b> $\downarrow$ 41.30

Table 10: Performance of the 7M Transformer models on the WIKITEXT-103 dataset.

16. For other hyper-parameters and the optimizer, we follow the default ones in their implementation.

During inference, we can use more contexts to construct memory. We train with different segment lengths, i.e.,  $L = 512$  or  $L = 2048$ . For the model trained with  $L = 512$ , it can leverage a long-term memory of a size 6,144 during inference; for the model trained with  $L = 2048$ , it can leverage a long-term memory of a size 12,228.

## F Performance of the 7M model on WIKITEXT-103

We conduct our ablation studies and analyses in §6 with an 8-layer Transformer model due to the limited computation budget. The model consists of

Model	Dev ( $\downarrow$ )
TRIMELM <sub>ext</sub>	41.50
w/o BM25 batching	45.71
w/o back-prop to memory	45.15

Table 11: Ablation studies of using BM25 batching and enabling back-propagation to memory representations during training. The numbers are on the WIKITEXT-103 development set (7M models).

Model	$L = 3072$	$L = 512$	$L = 150$
Transformer	82.70	81.15	81.40
+ cont. cache	67.45	71.29	75.10
★ TRIMELM	54.89	63.22	71.82

Table 12: Performance on the WIKITEXT-103 development set (7M models). We vary the segment  $L$  here to study the effectiveness of using local memory.

Model	Top-1	Top-8	Top-64	Top-1024
kNN-LM	25.82	50.03	69.85	86.97
★ TRIMELM <sub>ext</sub>	<b>27.64</b>	<b>51.16</b>	<b>70.43</b>	<b>87.18</b>

Table 13: Retrieval performance on external memory of our model (7M) and kNN-LM (Khandelwal et al., 2020) on the WIKITEXT-103 development set. We report top- $K$  retrieval accuracy ( $K = 1, 8, 64, 1024$ ).



Frequency	> 10k	1k-10k	100-1k	10-100	$\leq 10$	avg
Transformer	3.35	4.11	13.63	30.46	240.39	18.04
kNN-LM + cont. cache	<b>3.14</b>	3.85	12.92	26.90	196.03	16.23
★ TRIMELM	3.47	4.15	13.57	28.05	198.33	17.10
★ TRIMELM <sub>long</sub>	3.43	4.13	13.62	27.89	194.89	17.01
★ TRIMELM <sub>ext</sub>	3.15	<b>3.84</b>	<b>12.50</b>	<b>25.41</b>	<b>171.61</b>	<b>15.51</b>

Table 14: Averaged perplexity in each frequency bucket on the WIKITEXT-103 development set (247M models).

$p$	0.0	0.1	0.5	0.9	1.0
Perplexity	54.33	49.85	45.08	41.50	41.86

Table 15: The performance of our model TRIMELM<sub>ext</sub> on the development set (WIKITEXT-103, 7M models). We disable the local memory with a probability of  $p$  during training.

7M parameters, 8 layers and 4 heads in each layer. The embedding dimension is 128 and the intermediate dimension of FFN is 512. The model takes a segment of 3072 tokens as input. We compare our approach with baselines on this model architecture. As shown in Table 10, our approach improves over the baselines by a large margin. This shows that modeling memory explicitly is essential when the model capacity is limited.

## G Additional Analysis

**Ablation study on TRIMELM<sub>ext</sub>** We study the importance of packing segments with high BM25 scores in the same training batch, as well as the effectiveness of enabling back-propagation to memory representations during training. As shown in Table 11, when we random batch training segments (instead of using BM25 scores), the perplexity increases to 45.71 (+4.21). Also, enabling back-propagation to memory is crucial for our approach — the performance is much worse if we disable it.

**Effectiveness of using local memory** We study the effectiveness of our model TRIMELM that uses only local memory with different segment lengths  $L$ . As shown in Table 12, our model significantly outperforms the baselines in all the settings. This suggests that our model can leverage local memory very effectively to improve performance.

**Retrieval performance on external memory** When external memory is used in our experiments, we perform nearest-neighbor search over the entire memory set  $\mathcal{M}_{\text{ext}}$  to retrieve the top  $K$  keys (we

use  $K = 1024$ ). Table 13 compares the retrieval accuracy of our approach and kNN-LM (Khandelwal et al., 2020) for different  $K$ . Our approach outperforms kNN-LM in terms of retrieval results; this explains how our final perplexity surpasses kNN-LM when incorporating external memory.

### Perplexity breakdown for different frequencies

We aim to understand which type of memories improves perplexity of tokens in different frequency groups. We group tokens into 5 buckets according to their frequency on the development set. Table 14 shows the results for different models. TRIMELM and TRIMELM<sub>long</sub> improve the perplexity of rare words (i.e., frequency  $\leq 1k$ ) while achieving similar or slightly worse results for frequent words compared to the Transformer baseline. TRIMELM<sub>ext</sub> improves perplexity in all the buckets. Interestingly, kNN-LM with continuous cache does not perform significantly better compared to TRIMELM and TRIMELM<sub>long</sub> although these two models do not use external memory. This suggests that jointly training memory representations and the language model particularly help improve the performance of rare words.

## H Tuning $p$ for training with external memory

When training the model with local and external memory, to avoid the model to only relies on high-quality local memory, we disable the local memory with a probability of  $p$ . Here we study how  $p$  will affect the final performance of our model. The results of using different  $p$  are shown in Table 15. We find that when  $p = 0$ , the model performs poorly with external memory as the model learns to only leverage local memory and ignores external memory during training. By increasing  $p$ , this issue is mitigated. We set  $p = 0.9$  in our main experiments.