

---

# Block-Recurrent Transformers

---

**DeLesley Hutchins<sup>\*1</sup>, Imanol Schlag<sup>\*3†</sup>, Yuhuai Wu<sup>1</sup>, Ethan Dyer<sup>2</sup>, Behnam Neyshabur<sup>2</sup>**

<sup>1</sup> Google Research    <sup>2</sup> Google Research, Blueshift Team

<sup>3</sup> The Swiss AI Lab IDSIA, SUPSI & USI

{delesley, yuhuai, edyer, neyshabur}@google.com    imanol@idsia.ch

## Abstract

We introduce the Block-Recurrent Transformer, which applies a transformer layer in a recurrent fashion along a sequence, and has linear complexity with respect to sequence length. Our recurrent cell operates on blocks of tokens rather than single tokens during training, and leverages parallel computation within a block in order to make efficient use of accelerator hardware. The cell itself is strikingly simple. It is merely a transformer layer: it uses self-attention and cross-attention to efficiently compute a recurrent function over a large set of state vectors and tokens. Our design was inspired in part by LSTM cells, and it uses LSTM-style gates, but it scales the typical LSTM cell up by several orders of magnitude. Our implementation of recurrence has the same cost in both computation time and parameter count as a conventional transformer layer, but offers dramatically improved perplexity in language modeling tasks over very long sequences. Our model out-performs a long-range Transformer XL baseline by a wide margin, while running twice as fast. We demonstrate its effectiveness on PG19 (books), arXiv papers, and GitHub source code. Our code has been released as open source [\[1\]](#).

## 1 Introduction

Transformers have mostly replaced recurrent neural networks (RNNs), such as LSTMs [\[2\]](#), on tasks that involve sequential data, especially natural language. There are several reasons for their success. First, transformers process all elements of the sequence in parallel, and are thus faster to train on modern accelerator hardware. In contrast, an RNN must process tokens sequentially, which leads to slow step times during training, and large batch sizes in order to fully saturate GPUs or TPUs.

Second, an RNN must summarize and compress the entire previous sequence into a single state vector which is passed from one token to the next. The size of the state vector limits the amount of information that the RNN can encode about the previous tokens in the sequence. In contrast, a transformer can attend directly to past tokens, and does not suffer from this limitation.

Third, attention operates effectively over longer distances. The forget gate in an LSTM discards information moving forward, and causes vanishing gradients during backpropagation. In practice, this means that LSTMs struggle to send a clear signal over more than a few hundred tokens, far less than the typical size of the attention window in a transformer [\[3\]](#).

Despite these advantages, transformers also have a disadvantage. The computational complexity of self-attention is quadratic with respect to the sequence length, which is a limiting factor when attempting to process long documents, such as books, technical articles, or source code repositories. Moreover, a transformer has no memory of past context; any tokens that it cannot attend to are “invisible” to the model.

---

<sup>\*</sup>Equal Contribution    <sup>†</sup>Work done partially while interning at Google Research (Blueshift Team) and partially funded by ERC Advanced grant no: 742870 to J.Schmidhuber.

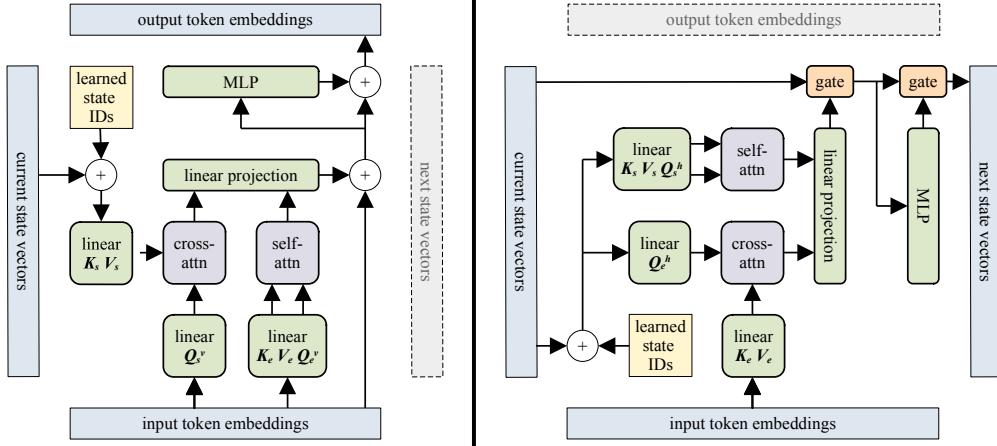


Figure 1: Illustration of our recurrent cell. The left side depicts the vertical direction (layers stacked in the usual way) and the right side depicts the horizontal direction (recurrence). Notice that the horizontal direction merely rotates a conventional transformer layer by  $90^\circ$ , and replaces the residual connections with gates.

In this work, we describe an architecture which combines the benefits of attention and recurrence. Like previous implementations of recurrence, our architecture constructs and maintains a fixed-size state, which summarizes the sequence that the model has seen thus far. However, our implementation of recurrence differs from previous work in several important aspects which together address the three limitations mentioned above.

Instead of processing the sequence one token at a time, **our recurrent cell operates on *blocks* of tokens**; see Figure 1. Within a block, all tokens are processed in parallel, at least during training. The recurrent cell likewise **operates on a *block* of state vectors rather than a single vector**. This means that the size of the recurrent state is orders of magnitude larger than in an LSTM, which dramatically improves the model’s capacity to capture the past. Processing the sequence in blocks also helps propagate information and gradients over longer distances, because the number of recurrent steps (and thus the number of times that the forget gate is applied) is orders of magnitude smaller. We show that the Block-Recurrent Transformer can remember information over distances of 60k tokens or more.

The recurrent cell itself is strikingly simple. For the most part, it consists of an ordinary transformer layer applied in a recurrent fashion along the sequence length. There are **a few tricks that are necessary to stabilize training**; see Sections 3.2 and 3.4 for details. The cost of recurrence, in terms of both computation time and parameter count, is essentially the same as simply adding one more layer to our transformer baseline. We demonstrate empirically that adding a single recurrent layer results in a much larger improvement in perplexity on multiple datasets than adding a conventional transformer layer, while training time and memory use are equivalent. Moreover, our recurrent cell is very easy to implement because it largely makes use of existing transformer code. Thus, our technique is a cheap and cheerful way to improve language modeling perplexity on long sequences.

## 2 Related Work

The quadratic cost of attention is well known in the literature, and a great deal of work has been done on efficient long-range attention mechanisms; see [4, 5] for recent surveys. Sparse strategies such as Big Bird [6], Routing Transformers [7], and Reformer [8] select only a subset of tokens to attend to. Hierarchical mechanisms [9] combine multiple tokens into phrases or sentences to reduce sequence length. Expire-span [10] learns to prune far-away tokens that the model has labelled as “unimportant”. Memorizing transformers [11] replace dense attention with  $k$ -nearest-neighbor lookup.

Yet another approach is to reduce the sequence length by pooling, averaging, or compressing it in some way. Hierarchical 1D attention [12], and Combiner [13] apply pooling or averaging over tokens at longer distances. Linformer [14] applies a linear transformation to the key and value matrices to reduce the sequence length. Compressive transformers [15] and funnel transformers [16] apply additional learned compression layers to compress the sequence.

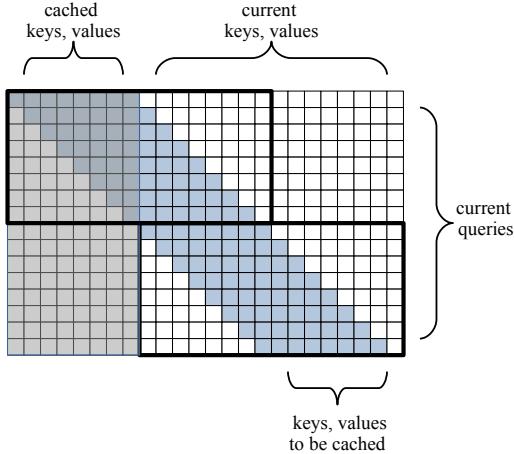


Figure 2: Sliding window, where segment length  $N = 16$ , window/block size  $W = 8$ . Keys and values for the first  $W$  shaded tokens were computed and cached on the previous training step; the remaining  $N$  unshaded tokens are the segment for the current training step. Instead of a single  $N \times (W + N)$  attention matrix, attention is done in two tiles of size  $W \times 2W$ .

The equation for attention is (roughly)  $\text{softmax}(\mathbf{Q}\mathbf{K}^T)\mathbf{V}$  where  $\mathbf{Q}$ ,  $\mathbf{K}$ , and  $\mathbf{V}$  are the query, key, and value matrices of the attention layer. If the softmax operation is removed from this equation or somehow “linearized”, the equation can be rearranged as  $\mathbf{Q}(\mathbf{K}^T\mathbf{V})$ , where  $(\mathbf{K}^T\mathbf{V})$  can be computed incrementally (i.e., in a recurrent fashion) as a cumulative sum over the sequence [17]. Linearized attention thus has linear rather than quadratic complexity with respect to sequence length. Following this line of reasoning, there have been several proposals that approximate the softmax [18] [19] or replace it [20] [21]. Linear transformers are related to earlier work on fast weight programmers [20] [22], and can be extended with other forms of recurrence [23].

Our work differs from all of the above mechanisms, because we rely only on standard dense attention with softmax.

A few other lines of research have combined the transformer architecture with recurrence in some way. The feedback transformer [24] allows lower layers to attend to the output of the topmost layer. Feedback has minimal cost at inference time, but it is unfortunately very slow to train because tokens must be processed sequentially. Simple Recurrent Units [25] [26] use a recurrence function that does not involve matrix multiplication, and is consequently much faster. RNMT+ combines RNNs and transformers in an encoder/decoder architecture to improve on translation tasks [27]. “Sandwich models” alternate between transformer and RNN layers and out-perform both transformers and RNNs on tasks involving source code [28]. The R-Transformer introduces an additional local RNN which can be computed in parallel in order to better model sequential structure [29]. The Perceiver architecture [30] is somewhat similar to ours; it also applies a transformer layer in an iterative fashion.

To the best of our knowledge, the idea of performing recurrence on blocks of tokens is underexplored. In the context of translation, [31] operates on sentences rather than tokens. Staircase Attention [32] also operates on blocks of tokens; each layer takes, as input, the outputs of the same layer from the previous block.

### 3 Method

The Block-Recurrent Transformer is based on sliding-window attention [33], which is an extension of ideas from Transformer-XL [34].

A long document, such as a book, consists of a *sequence* of tokens. Due to memory limitations, it is usually not possible to fit the entire sequence into device memory. Thus, the sequence is divided into *segments* of length  $N$  ( $N = 4096$  in our experiments), which are processed sequentially over a number of training steps. Each training step processes one segment.

The sliding window attention pattern is illustrated in Figure 2. Given a segment of  $N$  tokens, the sliding window applies a causal mask in which each token can only attend to the  $W$  previous tokens,

where  $W$  is the *window size* ( $W = 512$  in our experiments). Because of the causal mask, most entries of the  $N \times N$  attention matrix are masked out (assuming that  $W \ll N$ ). Thus, the attention computation can be optimized by breaking it into smaller tiles along the diagonal. The segment of  $N$  tokens is subdivided into *blocks* of size  $W$ , and each block attends locally to itself and to the previous block, so the size of each local attention matrix is  $W \times 2W$ . Using this mechanism, attention is quadratic with respect to the window size  $W$ , but linear with respect to the segment length  $N$ .

Borrowing an idea from Transformer-XL, the keys and values from the last block in each segment are stored in a non-differentiable *cache* for use on the next training step. By using the cache, the first block in the next segment can attend to the last block in the previous segment, which extends the sliding window to cover the entire (book-length) sequence. The cache implements a form of truncated backpropagation through time [35] over long documents.

Note that if  $N = W$ , then sliding window attention will behave exactly like Transformer-XL; it will process and cache one segment (i.e. one block) per training step. Setting  $N \gg W$  does not change the context length of attention, but it allows gradients to backpropagate across multiple blocks during training; we show that the improved differentiability provides a modest benefit to perplexity over Transformer-XL. See Appendix A for more details.

### 3.1 Recurrent Cell

A Block-Recurrent Transformer layer extends the sliding-window attention mechanism by adding a set of recurrent states, which are updated at the end of each block of  $W$  tokens. Our design for the recurrent cell is illustrated in Figure I, which depicts the operations done within a single block of the input sequence.

The recurrent cell receives two tensors as inputs: a set of  $W$  token embeddings, where  $W$  is the block/window size, and a set of  $S$  “current state” vectors. The cell produces two tensors as outputs: a set of  $W$  output embeddings, as well as a set of  $S$  “next state” vectors. We denote the function going from input token embeddings to output token embeddings as the *vertical* direction, and the function going from the current state vectors to the next state vectors as the *horizontal* direction. The number of state vectors  $S$  and the window size  $W$  are independent hyperparameters, but we set  $S = W = 512$  in our experiments to simplify comparisons against baselines.

The **vertical direction** of the cell is an ordinary transformer layer with an additional cross-attention operation, much like a decoder layer in a standard encoder-decoder architecture [36]. It does self-attention over the input tokens, and cross-attends to the recurrent states. Unlike a typical decoder layer, we do self-attention and cross-attention in parallel. The results of both forms of attention are concatenated together and fed into a linear projection.

The **horizontal direction** of the cell mirrors the forward direction, except that it performs self-attention over the current state vectors, and cross-attends to the input tokens. The recurrent direction also replaces the residual connections with gates, which allows the model to “forget”, an ability that is important for algorithmic tasks [37], or when processing long documents, where it has been central to the success of LSTMs [38].

Note that the presence of gates is the reason why self-attention and cross-attention are done in parallel. Doing them sequentially, as is standard practice, would introduce a third gate in the horizontal direction, which led to worse perplexity in our experiments.

Recurrence is integrated with the sliding window attention mechanism. Although not shown in Figure I, each cell also receives keys and values from the previous block as input, these are concatenated with  $(\mathbf{K}_e, \mathbf{V}_e)$  from the current block in order to implement sliding-window attention.

A Block-Recurrent Transformer *layer* processes the blocks within a segment sequentially by stacking recurrent cells horizontally, with the “next states” output of the previous cell feeding into the “current states” input of the next cell. In code, this is implemented as a simple for-loop over blocks. Multiple layers can also be stacked vertically in the usual fashion. Our experiments use a single recurrent layer, sandwiched between a number of non-recurrent layers that use sliding-window attention.

The final set of state vectors from the last block in the segment are cached, along with the keys and values, and used as the initial state for the first block on the next training step. Every layer in the stack (both recurrent and non-recurrent) has its own cache.

**Sharing of keys and values.** Keys and values are shared between the vertical and horizontal directions. One set of keys and values ( $\mathbf{K}_e, \mathbf{V}_e$ ) are computed from the input token embeddings, and another set of keys and values ( $\mathbf{K}_s, \mathbf{V}_s$ ) are computed from the recurrent state vectors. Queries are not shared, so there are four separate sets of queries:  $\mathbf{Q}_e^v$  and  $\mathbf{Q}_s^v$  in the vertical direction, and  $\mathbf{Q}_s^h$  and  $\mathbf{Q}_e^h$  in the horizontal direction.

### 3.2 State IDs and Position Bias

With a large number of state vectors, the total size of the recurrent state is far larger than that of an LSTM. However, the same weights (projection matrices and MLP) are applied to each state vector. Without some way to differentiate the states, the model will compute the same result for each state vector, thus negating any advantage from having multiple states. To prevent this failure mode, we add a set of learned “state IDs” to the state vectors before computing the keys, values, and queries. These “state IDs” allow each state vector to consistently issue different queries against the input sequence, and against other states. State IDs are identical to learned position embeddings; we use a different name because there’s no notion of “position” between states.

We do not add global position embeddings to the tokens, because global position embeddings don’t work well for long sequences [34]. Instead, we add a T5-style relative position bias [39] to the self-attention matrix in the vertical direction. (Although similar, T5 relative positions differ slightly from the relative positions used in the Transformer-XL paper [34].) When the recurrent states cross-attend to input tokens, there is no position bias, because the relative distance between “state” and “token” is undefined.

We also normalize queries and keys as described in [40]; we found that normalization improved the stability of Transformer-XL when used with a relative position bias.

### 3.3 Gate Type

We experimented with two different gating mechanisms for the recurrent cell. Each state vector has its own gate, but all state vectors are updated in parallel, using the equations below.

**Fixed gate.** The fixed gate uses a learned convex combination, similar to highway networks [41].

$$\mathbf{z}_t = \mathbf{W}_z \mathbf{h}_t + \mathbf{b}_z \quad (1)$$

$$\mathbf{g} = \sigma(\mathbf{b}_g) \quad (2)$$

$$\mathbf{c}_{t+1} = \mathbf{c}_t \odot \mathbf{g} + \mathbf{z}_t \odot (1 - \mathbf{g}) \quad (3)$$

where  $\mathbf{W}_z$  is a trainable weight matrix,  $\mathbf{b}_z$  and  $\mathbf{b}_g$  are trainable bias vectors,  $\sigma$  is the sigmoid function,  $\mathbf{c}_t$  is the cell state for the current block (i.e., the state for the block at index  $t$  in the sequence of blocks),  $\odot$  is the element-wise multiplication, and  $\mathbf{h}_t$  is the current input to the gate. In our model,  $\mathbf{h}_t$  is either the output of attention, in which case  $\mathbf{W}_z$  is the linear projection that feeds into the gate, or  $\mathbf{h}_t$  is the output of the hidden layer of the MLP, in which case  $\mathbf{W}_z$  is the final layer of the MLP.

Unlike highway networks, the bias  $\mathbf{b}_g$  is a simple learned vector of shape  $\mathbb{R}^d$ , which is broadcast over all state vectors, where  $d$  is the state embedding dimension. The value of  $\mathbf{g}$  does *not* depend on either the current value of the state vector  $\mathbf{c}_t$ , or on the current input  $\mathbf{h}_t$ , and thus remains constant (i.e., fixed) after training. The fixed gate essentially implements an exponential moving average over previous blocks.

**LSTM gate.** The LSTM gate uses the standard combination of input and forget gates:

$$\mathbf{z}_t = \tanh(\mathbf{W}_z \mathbf{h}_t + \mathbf{b}_z) \quad (4)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{h}_t + \mathbf{b}_i - 1) \quad (5)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{h}_t + \mathbf{b}_f + 1) \quad (6)$$

$$\mathbf{c}_{t+1} = \mathbf{c}_t \odot \mathbf{f}_t + \mathbf{z}_t \odot \mathbf{i}_t \quad (7)$$

where  $\mathbf{W}_z, \mathbf{W}_i, \mathbf{W}_f$  are trainable weight matrices, and  $\mathbf{b}_z, \mathbf{b}_i, \mathbf{b}_f$  are trainable bias vectors. The LSTM gate is strictly more expressive, because the values of  $\mathbf{f}_t$  and  $\mathbf{i}_t$  depend on the current input  $\mathbf{h}_t$ . In our model,  $\mathbf{h}_t$  depends on  $\mathbf{c}_t$ , so the LSTM gate also depends indirectly on  $\mathbf{c}_t$ . LSTM gate values are thus different for each state vector, and for each block index  $t$ .

### 3.4 Gate Initialization and Training Stability

We observed that training stability is quite sensitive to how the gates are initialized. Recurrence has a failure mode where the model learns to completely ignore the recurrent state, in which case its performance reverts to that of the non-recurrent transformer. Moreover, this situation appears to be a local optimum; once the model has reached this point, it does not recover. We stabilize training by initializing the weights and bias to small but non-zero values, and adding a constant -1 and +1 to the input and forget gates to bias them to “remember”. See Appendix B for details.

### 3.5 Gate Configuration

We experimented with three different gate configurations.

**Dual.** The dual gate configuration is the one shown in Figure I, in which both of the residual connections in the cell are replaced with gates. The disadvantage of this configuration is that there are two gates, both of which can forget.

**Single.** The single gate configuration removes the linear projection and the gate that is attached to it. Instead, the concatenation of self-attention and cross-attention is fed directly into the MLP.

**Skip.** The skip configuration removes the MLP and the gate that is attached to it. This configuration is similar to the single-gate version, except that it is strictly weaker. Instead of a two layer MLP with a very large hidden layer, it uses a linear projection with no nonlinearity.

### 3.6 Placement of Recurrence and Computation Cost

**Single recurrent layer.** The basic version of the Block-Recurrent Transformer uses a single recurrent layer sandwiched between a number of non-recurrent transformer layers with sliding attention. We use a 12-layer model with recurrence on layer 10. All layers have a Transformer-XL-style cache.

**Cost of recurrence.** During training, the 12-layer Block-Recurrent Transformer has almost exactly the same computation cost, in both parameters and FLOPS, as a 13-layer Transformer-XL model without recurrence. The two are equivalent because the recurrent cell does almost the same operations as a conventional transformer layer, merely in the horizontal instead of the vertical direction.

The inference cost for autoregressive decoding is also nearly identical, for the same reason. Recurrence adds an additional attention operation per token, the cost of which is the same as self-attention in a 13th layer.

## 4 Results

We tested the Block-Recurrent Transformer on three different data sets of long documents: PG19, arXiv, and GitHub. The PG19 dataset [42] contains full-length books written prior to 1919 from project Gutenberg. The arXiv dataset [11] is a corpus of technical papers downloaded via the arXiv Bulk Data Access<sup>1</sup> and filtered to include only articles labeled as “Mathematics” and whose L<sup>A</sup>T<sub>E</sub>X source is available. The GitHub dataset [11] is a corpus of source code from different GitHub repositories with open-source licenses. All of the files in each GitHub repository are concatenated together to make one long document.

The task is auto-regressive language modeling, where the goal is to predict the next token in the sequence. We report bits-per-token numbers (i.e.  $\log_2$  perplexity; lower is better) for all models. Further training details for each dataset can be found in Appendix C.

### 4.1 Baselines

We compare the Block-Recurrent Transformer to five different baselines. The first baseline, XL:512, establishes a reference point against which various other improvements can be compared. It’s a

<sup>1</sup>[https://arxiv.com/help/bulk\\_data](https://arxiv.com/help/bulk_data)

Table 1: Average bits-per-token ( $\log_2$  perplexity) of each model. The recurrent models (named `Rec:gate:config`) have the same computational cost as the `Slide:13L` baseline, but much better perplexity. They even outperform the `XL:2048` baseline, **while running more than twice as fast**. Measured error bars on PG19 are low, between 0.002 and 0.007, but are rounded up to 0.01 to match the precision of results in the table. Step time is for a single training step (lower is better). For PG19, we train both character-level (bytes) and token-level models.

Model	segment length	window length	step time (relative)	PG19 bytes	PG19 tokens	arXiv tokens	GitHub tokens
<code>XL:512</code>	512	512	0.88	1.01	$3.62 \pm 0.01$	1.45	1.21
<code>XL:1024</code>	1024	1024	1.20	0.997	$3.59 \pm 0.01$	1.37	1.08
<code>XL:2048</code>	2048	2048	2.11	0.990	$3.58 \pm 0.01$	1.31	1.01
<code>Slide:12L</code>	4096	512	0.93	0.989	3.60	1.43	1.19
<code>Slide:13L</code>			1.00	0.989	$3.58 \pm 0.01$	1.42	1.17
<code>Rec:lstm:dual</code>	4096	512	1.06	0.985	$3.54 \pm 0.01$	1.26	1.01
<code>Rec:lstm:single</code>			1.05	0.962	$3.54 \pm 0.01$	1.29	1.03
<code>Rec:lstm:skip</code>			1.00	0.969	$3.56 \pm 0.01$	1.31	1.10
<code>Rec:fixed:dual</code>			1.01	0.957	<b><math>3.52 \pm 0.01</math></b>	1.27	0.991
<code>Rec:fixed:single</code>			1.02	0.966	$3.58 \pm 0.01$	1.25	1.00
<code>Rec:fixed:skip</code>			<b>0.99</b>	<b>0.952</b>	$3.53 \pm 0.01$	<b>1.24</b>	<b>0.976</b>
<code>Feedback:lstm:single</code>	4096	512	1.40	0.977	3.50	<b>1.22</b>	-
<code>Feedback:fixed:skip</code>			1.35	<b>0.935</b>	<b>3.49</b>	1.24	-
Memorizing Trans. 64k	512	512	1.94	0.950	3.53	<b>1.22</b>	-

12-layer Transformer-XL model with a window size of 512, and 150 million parameters. It has 8 heads of size 128, embedding vectors of size 1024, an MLP with a hidden layer of size 4096, and the relu nonlinearity. It uses a Transformer-XL style cache, but no sliding window, so the segment length is the same as the window size, i.e., it is trained on segments of 512 tokens.

`XL:1024` and `XL:2048` are similar, but have window sizes of 1024 and 2048, respectively. As expected, increasing the window size improves perplexity, especially on the arXiv data set. However, these two models still have worse perplexity than the recurrent model, as well as being much slower.

`Slide:12L` is a 12-layer transformer with a window size of 512, but uses a sliding window over a segment of 4096 tokens. This model is almost identical to `XL:512`; the only difference is that the sliding window is differentiable over multiple blocks, while the Transformer-XL cache is not.

`Slide:13L` adds a 13th layer, and is directly comparable to the recurrent models in terms of both computation cost (FLOPS or step-time), number of parameters, and segment length. Notice that adding another layer with more parameters yields a much smaller improvement than adding recurrence.

**Relative cost.** All five baselines, and all 6 recurrent models, have roughly the same number of parameters: between 151 million (12 layer) and 164 million (13 layer or recurrent). The training speed (i.e. step time) of each model is shown in Table 1 (lower is better). Because the raw step time depends on hardware and compiler, we report numbers relative to the `Slide:13L` baseline.

**Batch Size.** We adjust the batch size so that each model processes the same number of tokens (and thus the same amount of training data) per training step. Thus, `XL:512` (segment length 512) runs at a batch size of 256 (8 per replica), while `Slide:12L` (segment length 4096) runs at a batch size of 32 (1 per replica) on PG19.

## 4.2 Benefit of Recurrence

We compare the 5 baselines to all six gate configurations for the Block-Recurrent Transformer. The recurrent model reliably outperforms all five baselines. The best overall configuration is `Rec:fixed:skip`, which outperforms the others in 3 out of 4 cases, and comes within the margin of error in the remaining case. This is especially notable because it is also the fastest configuration, having a slightly *lower* step time and fewer parameters than `Slide:13L`, because it does not have

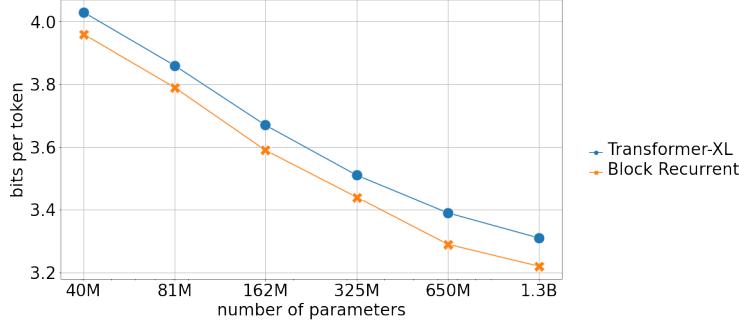


Figure 3: Scaling of the 12-layer Block-Recurrent Transformer vs 13-layer Transformer-XL on PG19. FLOPs are the same between the two models at a given parameter count. **At larger sizes, adding recurrence is equivalent to doubling the number of parameters.** Details in Appendix F.

the MLP. It is better than the 13-layer baseline by a wide margin, and it is even better than the Transformer-XL model with a window size of 2048, which runs over 2 times slower.

The other gate configurations also outperform the 13-layer baseline, but their relative ranking varies according to the dataset. Despite being theoretically more powerful, the LSTM gate tends to lag behind the fixed gate in all of our experiments.

**Scaling up.** Figure 3 shows the effect of adding recurrence as the transformer model is scaled up and down in size. We trained six different models on PG19, ranging in size from 40M parameters to 1.3B parameters. For the four smaller models, we compare a 12-layer Block-Recurrent Transformer against a 13-layer Transformer-XL baseline, while for the two larger models, we compare a 24-layer Block-Recurrent Transformer, with recurrence at layers 10 and 20, against a 26-layer Transformer-XL baseline. This experiment used a cosine-decay learning rate as described in [43], and a custom 32k SentencePiece vocabulary [44]. More details are in Appendix F.

Our experiments show that recurrence provides a consistent benefit across all scales. The relative improvement actually seems to increase with the number of parameters; at larger sizes **recurrence provides a benefit which is greater than doubling the number of parameters.**

### 4.3 Ablations

**Multiple recurrent layers.** Adding two recurrent layers right next to each other in the stack (layers 9 and 10) did not improve model perplexity. Adding two layers widely separated in the stack (layers 4 and 10) did provide an improvement, but the improvement was no better than simply adding another non-recurrent layer to the stack. Previous work on Memorizing Transformers [11] showed a similar effect. In our qualitative study, we saw that the model seems to use recurrence primary for long-range name lookups, much like memory. We conclude that one layer of recurrence is sufficient for the model to extract most of the benefits, although we did use two layers for our largest models.

**Number of recurrent state vectors.** We trained the model with differing numbers of state vectors, from 128 to 2048. Increasing the number of states makes a small but measurable improvement up to 1024, but the model does worse with 2048 (see Appendix D). We hypothesize that the model has trouble learning to use the recurrent state effectively if the state space grows too large.

**Reducing window size.** Reducing the size of the sliding window dramatically reduces perplexity for Transformer-XL, because it reduces the amount of context that the transformer is able to attend to. Reducing the size of the window in a recurrent transformer has a smaller effect, because the model can use recurrence to compensate (see Appendix D).

### 4.4 Block feedback

Inspired by the feedback transformer [24], which allows all layers to attend to the topmost layer, we implemented a variation in which every layer of the transformer (not just the recurrent one) can cross-attend to the state vectors in the recurrent layer. This variation further improves perplexity, but

Table 2: Comparison with other published work on PG19. Fields marked - are unknown.

Model	Layers	perplexity word-level	parameters	vocabulary size
Compressive Transformer [15]	36	33.6	-	32k
Routing Transformer [7]	22	33.2	490M <sup>1</sup>	98k
Perceiver AR [45]	60	28.9	974.6M <sup>1</sup>	32k
Block-Recurrent Transformer	24	28.46	650M	32k
Block-Recurrent Transformer	24	<b>26.50</b>	1.3B	32k

at a cost; step time increased by approximately 35-40%, and the additional queries also increase the number of parameters. Results are shown in Table I, and further described in Appendix E.

#### 4.5 Comparisons against prior published work

The PG19 test set contains 6,966,499 words [15], which are broken into 10,229,476 tokens using a SentencePiece vocabulary, trained on PG19. Our 24-layer 1.3B parameter model achieves 3.22 bits per token, and thus **achieves a new state of the art word-level perplexity of 26.50** (Table 2). However, we note that raw perplexity numbers are not necessarily a meaningful way to compare architectures, because they depend on numerous other factors, such as the number of parameters, vocabulary, learning rate schedule, batch size, etc.; a more detailed discussion is in Appendix C.3.

We were able to run a fair comparison (identical vocabulary, configuration, and hyperparameters) of the Block-Recurrent Transformer against the Memorizing Transformer [11], with a memory of size 64k (Table I). The memorizing transformer is constructed similarly to our model; it has one layer which has been augmented with a mechanism that gives it the ability to attend over much longer distances. We find that Block-Recurrent does almost as well as the Memorizing Transformer on arXiv, and does just as well on PG19, but trains almost twice as fast. However, there are many ways of implementing approximate  $k$ -nearest-neighbor lookup, so relative speed will be highly implementation-dependent; our implementation runs on TPU, and does not use custom CUDA kernels.

#### 4.6 Qualitative analysis

Prior work on long-context transformers [42] [11] has found that attention at long ranges is typically used to look up proper names, such as characters or places. We performed a qualitative analysis in an attempt to determine whether our model is using recurrence in the same way. We selected 5 books at random from the PG19 test set, ran both the Block-Recurrent Transformer and the 13-layer Transformer-XL on each book, and then compared the cross-entropy loss for all tokens. We sorted the results, and examined the top 4 tokens from each book with the greatest difference: the tokens for which the predictions of the recurrent model have the largest improvement over the baseline.

In 17/20 cases, the recurrent model predicted a proper name, usually with relatively high probability, that Transformer-XL was unable to predict. In 2 cases it predicted a chapter title (having previously seen the table of contents), and in the last case, it predicted a foreign-language word that was unique to that book. In 19/20 cases, the predicted word was nowhere within the attention window, so it must have been stored within the recurrent state (details in the appendix, Section G).

In a second study, we compared the recurrent model, running normally, against a variation in which the recurrent state is cleared at the end of each 4096-token segment, instead of being cached. Clearing the state degrades the model’s ability to predict dependencies at a longer range than the segment length; typical mispredictions once again included proper names and chapter titles. Interestingly, this study also showed that the recurrent model is able to remember the title and author of a book (which is part of the Gutenberg boilerplate at the beginning and end of each book) **across the entire length of the book – more than 60,000 tokens**. See Appendix G.1

A further quantitative comparison of the per-token cross-entropy between Transformer-XL and the Block-Recurrent Transformer is given in Appendix H.

<sup>1</sup>Personal communication.

## 5 Discussion

Our implementation of recurrence was inspired by the way that humans seem to process long sequences. When a human reads a novel, they do not attempt to remember every single word in the book. Instead, a human reader will construct a mental model, or knowledge graph, which summarizes the story thus far, i.e., the names of the main characters, the relationships between them, and any major plot points. When a human reads a paragraph of text, they will parse the information in the paragraph, process and interpret the information using background knowledge from their mental model, and finally update their mental model with new information. Our recurrent architecture loosely mimics this process. It takes a block of text, and parses it by running it through a conventional transformer stack. Tokens in the text attend to the recurrent states (i.e. the mental model), and the states, in turn, are updated by attending to the text.

Based on our qualitative analysis, it seems that the model is, in fact, using the recurrent state to summarize some of the information about frequently occurring characters and places. However, it does not seem to be doing much complex reasoning, as evidenced by the fact that our best performing model is the `fixed:skip` configuration. This configuration does not use a complex LSTM-style gate, which chooses to remember or forget based on its current state and inputs; instead, it simply computes an exponential moving average, not unlike some other forms of long-range approximate attention.

Moreover, the `skip` configuration cuts out the large MLP from the recurrent transformer layer. In a vanilla transformer, removing the MLP from all layers would severely degrade the model [46]; those large MLPs are computing something important. In a recurrent layer, removing the MLP makes little difference; it does not seem to be computing anything useful. We conclude that training the recurrent layer to make full use of its capabilities for knowledge extraction and summarization will require further advances.

### 5.1 Ethics

The potential negative social impacts from this work are similar to any other advance in language modelling. Large language models could potentially be used to create disinformation and fake news, power malicious chatbots, or generate spam. The Block-Recurrent Transformer can potentially create longer documents than was previously feasible, thus expanding the range of applications in which these negative impacts could occur. The best way to mitigate these risks is to train models that can reason about text, and flag misinformation or malicious content.

## 6 Conclusion

We have shown that when training language models on long documents, the Block-Recurrent Transformer provides a greater benefit at lower cost than scaling up the transformer model in other ways. Adding recurrence to a single layer has roughly the same cost as adding an additional non-recurrent layer, but results in a much larger improvement to perplexity. We have also shown that recurrence provides a larger benefit than simply increasing the window size of attention, or increasing the number of parameters. **Our medium-sized model has lower perplexity than a Transformer-XL model with 4 times the window size, but runs twice as fast, and our larger model outperforms a Transformer-XL model with twice the number of parameters.**

Furthermore, in contrast to some other recently proposed transformer variants, the Recurrent Transformer is very easy to implement, since it consists mostly of ordinary transformer components and RNN gates. No custom CUDA kernels are required. Our code has been released as open source [1].

Evaluating block-recurrent transformers on downstream tasks is an important direction for future work. We believe that the Block-Recurrent Transformer will be most useful in situations that require long-range context; examples of potential applications include writing book reports, summarizing long news articles, code completion, or question/answering over book-length works. There are a number of new and emerging benchmarks that test long-range performance [47, 48, 4]. Previous studies have found a strong correlation between language modeling and diverse downstream tasks [49, 50].

Despite our initial successes, we also believe that the recurrent architecture that we present here has not yet achieved its full potential, and there are opportunities for future research and further improvements in this area.

# Appendices

## Appendix A Further Analysis.

The *context length* of an autoregressive language model refers to the number of previous tokens that the model can make use of when predicting the next token. A vanilla transformer operating on segments of length  $N$  has a context length of 0 for the first token, and  $N - 1$  for the last token, and thus has an average context length of  $N/2$ . The prediction quality for a vanilla transformer is not uniform across the segment; it makes poor predictions for tokens near the beginning of the segment due to lack of context, and better predictions near the end.

With sliding window attention, each layer attends to the previous layer within a window of  $W$  tokens from the current position. The context length for a single layer is thus  $W$ , no matter where in the segment the token occurs. The simple case where  $W = N$  corresponds to Transformer-XL. A Transformer-XL model achieves a large improvement in average perplexity over a vanilla model, simply because it can make much better predictions for the tokens at the beginning of each segment.

In a model with multiple layers, the *theoretical receptive field* (TRF) is defined as the maximum distance that information could potentially propagate through the model. This definition is similar to “context length”, but the TRF is “theoretical”, because the model may have a hard time actually learning to use that much context in practice. For example, the TRF of an LSTM is infinite, but in practice an LSTM has difficulty transmitting information over more than a few hundred tokens. The *effective context length* of an LSTM is much less than the TRF might suggest.

For a sliding-window model (or Transformer-XL), the TRF is  $W \cdot L$ , where  $L$  is the number of layers. However, the model can still only attend directly to the previous  $W$  tokens. Although the TRF is much higher, making use of the additional context requires multiple “hops” of attention, which is more difficult for the model to learn. This problem is especially acute in Transformer-XL ( $N = W$ ), because the very first “hop” of attention will be to keys and values in the cache, which is not differentiable. Our sliding window model ( $N \gg W$ ) has an identical TRF to Transformer-XL, but it can differentiate across multiple blocks, which gives it a higher effective context length in practice, and thus better perplexity.

The TRF of the block-recurrent transformer is infinite. We also show that the effective context length also seems to be quite large in practice, since we observe cases in which the model is able to accurately predict information across distances of more than 60k tokens.

### A.1 Computational Complexity

The computational complexity of attention in a recurrent layer is  $\mathcal{O}((W^2 + S^2 + 2SW) \cdot N/W)$ , where  $N$  is the segment length,  $W$  is the window length, and  $S$  is the number of states.  $N/W$  is the number of blocks, and each block does self attention  $W^2$ , state self-attention  $S^2$ , and attention between tokens/states and states/tokens  $2SW$ .

The complexity of sliding-window attention is  $\mathcal{O}(W^2 \cdot N/W) = \mathcal{O}(W \cdot N)$ .

### A.2 Comparison between context, recurrence, and memory

The arXiv data set contains latex code, with lots of complicated syntax that can benefit from long-range attention (e.g. theorems, jargon, citations). As a result, adding memory, recurrence, or just increasing the window size of the Transformer-XL baseline yields a larger benefit on arXiv than it does for PG19.

There also seems to be a qualitative difference in how the models are using attention on these datasets. We hypothesize that dealing with complicated syntax requires direct (single-“hop”) attention, which is why the XL:2048 model does well on arXiv. In contrast, natural language novels don’t have complicated syntax, but may benefit from a better understanding of subtle relationships between words (e.g. which character the word “her” refers to). These subtle interactions require multi-layer, multi-“hop” attention, which is easier to train in the more differentiable Slide:13L model. On PG19, Slide:13L actually does better than XL:2048, despite having a much shorter window size, while on arXiv, the situation is reversed.

There may be a similar relationship between recurrence and  $k$ NN memory. The Memorizing Transformer does direct, single-“hop” attention, using  $k$ NN lookup. Recurrence, in contrast, summarizes and compresses text into a set of recurrent states.

Both the Memorizing Transformer and the Block-Recurrent Transformer achieve very similar perplexity, and based on qualitative studies, they seem to use the additional memory or states primarily for long-range name lookups. However, recurrence may be better at capturing more subtle long-range information, like writing style, while memory is better at precision lookups of facts, like citations. The fact that the Memorizing Transformer does better than the Block-Recurrent transformer on arXiv, but not PG19, would seem to support this hypothesis, but further experiments on downstream tasks are necessary to confirm this hypothesis.

### A.3 Comparison against Longformer

The idea of sliding window attention was popularized by the Longformer [33], but the full Longformer is a much more complicated model than the S1ide model that we present here.

The full Longformer uses several different attention patterns, and sliding-window attention is only one of them. Longformer also uses dilated attention and sparse global attention, both of which are implemented with custom CUDA kernels. Moreover, LongFormer uses different window sizes in each layer, and it uses a multi-phase training regimen of pre-training and fine-tuning, following a curriculum that gradually increases window size and sequence length. We do none of these things.

## Appendix B Gate Initialization and Training Stability

We observed that training stability is quite sensitive to how the gates are initialized. Recurrence has a failure mode where the model learns to completely ignore the recurrent state, in which case its performance reverts to that of the non-recurrent transformer. Moreover, this situation appears to be a local optimum; once the model has reached this point, it does not recover.

Our hypothesis is that learning the recurrent transition function is a much more difficult task than learning to attend directly to the input tokens. As a result, the vertical direction trains much faster than the horizontal direction, especially early in training. This may lead to a situation in which the recurrent states are much less informative than the input tokens, and the model learns to ignore them.

To avoid this failure mode, gate initialization requires special care. Moreover, proper gate initialization depends on the optimizer. We use the Adafactor optimizer [51], which normalizes gradients with respect to their variance, and then multiplies them by the native scale of the underlying parameter. Thus, if a bias term is initialized to 0, its native scale will be 0, the gradient updates will be very small, and the bias will tend to remain small over the course of training. If a bias term is initialized to 1 (which tells the forget gate to “remember”, and is standard practice in LSTMs) then the initial updates will be large, and the model will learn to ignore the recurrent states before they have the chance to learn anything useful.

We compromise by initializing the bias terms of the gates to small but non-zero values, using a normal distribution with mean 0 and a standard deviation of 0.1. The weight matrices of the gates are also initialized to small values, using a truncated normal distribution with a standard deviation of  $\sqrt{\frac{0.1}{f_{in}}}$  where  $f_{in}$  is the dimension of  $h_t$ .

We add a constant of -1 and +1 to the input and forget gates (see Eq. 4) to initially bias the gate to “remember” without affecting the size of the updates that Adafactor will apply. Using this initialization trick, the recurrent cell reliably learns to make use of the recurrent state.

## Appendix C Training Details

For PG19, we do both token-level modeling, and character-level modeling. In our initial experiments, we use a pre-trained 32k sentencepiece vocabulary from T5 [39] for the token-level modeling. We use the Adafactor optimizer [51], a learning rate schedule with inverse square root decay, 1000 warmup steps, and a dropout rate of 0.05. The learning rate is 1.0; when combined with warmup and the decay schedule, this yields an applied learning rate of 0.03, decaying to 0.0014. This learning rate

and schedule were borrowed from other language models; we did not attempt to do a hyperparameter sweep to identify the optimum learning rate and schedule. We train for 500k steps with 32 replicas on Google V4 TPUs; training takes approximately 48 hours. Reported results are for the "test" split.

For the later scaling experiment on PG19, we switched the learning rate schedule to cosine decay, as recommended in [45], with a maximum rate of 0.01, and a minimum of 0.001. We did a brief experiment with a learning rate of 0.02 and 0.005, before settling on 0.01. The change in learning rate schedule resulted in a significant improvement. We also switched from the 32k T5 vocabulary to a 32k custom sentencepiece vocabulary trained on PG19. Our custom vocabulary has higher bits-per-token, but fewer tokens, and thus has slightly better word-level perplexity.

For arXiv, we use a pre-trained 32k vocabulary from LaMDA [52]. Due to the large number of mathematical symbols in LaTeX, many tokens are only one character, so the bits-per-token numbers are lower than for PG19. We dropped the learning rate to 0.5 after observing some instabilities when training on longer (4096) segment lengths. Reported results are for the "test" split.

The GitHub dataset is much larger than PG19, and has very high variance, due to the fact that it contains code written in many different programming languages and coding styles. Consequently, there was a lot of noise in the results, which made it difficult to accurately compare models. We reduced the noise by using a batch size that is 4x larger than for PG19. As with the PG19 scaling study, we use a cosine decay learning rate schedule, but the maximum LR is 0.005; this is half the LR used for PG19. Because of the increased batch size, these models ran for only 250k steps. The GitHub experiment uses a pre-trained 32k vocabulary from LaMDA. Reported results are for the "validation" split.

Due to the large number of experiments, we did not have the computational resources to run all experiments multiple times, and consequently do not provide error bars for all experiments. However, for the headline numbers on PG19-tokens, we ran each experiment 3 times, with both different random seeds and with dataset shuffling. Actual measured error bars on PG19 were very low, between 0.002 and 0.007. The numbers in Table I are rounded to the nearest 0.01, which means that the error bars must be rounded up to match the precision of the reported results. E.g. for `Rec:fixed:skip` on PG19-tokens, an average of 3 runs has a mean of 3.525 and a standard deviation of 0.0047; we round this *up* to  $3.53 \pm 0.01$ . Note that it is not possible to obtain truly accurate error bars from such a small number of runs; by rounding the error up, we provide a conservative estimate of the actual error.

## C.1 Dataset licensing and other issues.

PG19 consists of works in the public domain, and consequently it is a public dataset that is freely available to other researchers. Due to the age of the texts, some of the books do contain potentially offensive material.

The arXiv dataset consists of documents for which the author has given express permission for their work to be distributed by [arxiv.org](http://arxiv.org). However, because the author still retains copyright, these articles cannot necessarily be redistributed in the form of a public dataset, nor will we publish a model that has been pre-trained on this data. We obtained access to this dataset via private channels.

The Github dataset consists of code with open-source licenses, which permit the code to be downloaded, compiled, or modified. Similar to arXiv, however, because the authors retain copyright, this code cannot necessarily be redistributed as a public dataset, nor will we publish a model that has been pre-trained on this data. We obtained access to this dataset via private channels.

## C.2 Selection of data sets

Having a standardized data set is import for the purpose of comparing published results, and historically, papers on long-context language modeling have used enwik8 or wiki-103 as benchmarks [34][33]. However, these datasets are not particularly good benchmarks for our purposes.

The purpose of our experiments is to see whether block recurrence can transmit information over very long distances: we show retrieval over 60k+ tokens. We chose PG19 specifically because we believe it to be a good dataset for these sorts of experiments. It consists only of long, book-length works, it is much larger than enwik8 or wiki-103, it is publicly available, and has been cited in other published work. Arxiv and github are (sadly) not public, but they similarly have long documents in the 50k+ token range.

Enwik8 is not a corpus of long articles. In fact, it doesn't even split the text into separate articles at all; it's just a single text dump that concatenates a bunch of short unrelated articles together. Attempting to split it on article boundaries yields a data set in which the majority of "articles" are merely stubs, with HTML boilerplate and no actual text. Enwik8 is a fine benchmark for data compression, which was the purpose for which it was originally intended, but it is less than ideal for long-range language modeling.

Wiki-103 is significantly better, because it does break the text into articles, and it eliminates the boilerplate, but the average length is still only 3.6k tokens per article, which is less than the segment length used in our experiments, and much shorter than the 50k-100k tokens of PG19.

### C.3 Comparisons with previously published results.

It is well-known that transformer perplexity scales with the number of parameters. However, the choice of vocabulary, learning rate schedule, batch size, number of training steps, and optimizer also make a large difference to the final headline perplexity numbers. The Chinchilla scaling study [43] demonstrates that a change to the learning rate schedule can have a large effect; we also observed improvements from a change to vocabulary as well. Not all vocabularies are created equal, even for vocabularies which have the same size, and are trained primarily on English-language text. Published perplexity numbers between different models cannot be meaningfully compared unless all other variables are strictly controlled.

## Appendix D Window Size and Number of Recurrent States

Ablation results for the window size is shown in Table 3(a). Decreasing window size leads to worse perplexity in both the recurrent model and Transformer-XL, but the penalty is smaller for the recurrent model.

Ablation results for the number of recurrent states is shown in Table 3(b). Increasing the number of recurrent states makes a small but measurable improvement up to 1024 states, but is worse at 2048 states. The window size was 512 for this experiment.

Table 3: Changing the window size (a) and number of recurrent states (b) on PG19.

Window size	Rec:fixed:skip	Slide:12L	Number of states	Rec:fixed:skip
128	3.58	3.69	128	3.54
256	3.54	3.64	256	3.535
512	3.53	3.60	512	3.53
			1024	3.51
			2048	3.55

## Appendix E Block-Feedback

In the block-feedback variation of our model, the entire stack of 12 layers is applied to the first block of tokens. The recurrent state for that block is then extracted from the recurrent layer, and the state is broadcast to all other layers when processing the next block. Because the recurrent layer is placed high in the stack, this means that the lower layers of the transformer can cross-attend to a higher layer, which is computationally more powerful, much like the feedback transformer [24]. Results are shown in Table 1.

Recurrence with feedback is significantly more expensive than the non-feedback version, because all 12 layers now have a cross-attention module, instead of just the recurrent layer. In our experiments, feedback increased the step time by approximately 35-40%, and the additional queries also increase the number of parameters.

Adding feedback improves perplexity in most cases, but the improvement seems to depend on the data set. The effect of feedback also depends on the gate configuration. In particular, block feedback dramatically improves the performance of the LSTM gate. This could be because the recurrent states, and thus the gate, get a gradient from all layers of the transformer, instead of just one.

## Appendix F Scaling Plot Details

Table 4: Bits per token on PG19 at various model scales. The data in this table was used for the scaling plot in Figure 3.

	40.6M	81.2M	162M	325M	650M	1.3B
Rec:fixed:skip	3.96	3.79	3.59	3.44	3.29	3.22
XL:512:13-layer	4.03	3.86	3.67	3.51	3.39	3.31

Performance on large text datasets, such as PG-19, is highly correlated with the number of trainable parameters; larger models tend to perform better. However, training large models can be expensive, and not all researchers have access to the necessary amount of compute to beat our new state of the art, or even to reproduce our results.

Table 4 provides the numeric results from our scaling study, which covers scales from small 40M parameter models that can be easily trained on a single machine, to our largest 1.3B parameter model. Configuration files for all scales are provided in the open source release. Our scaling strategy is to increase the dimensions of the various parts of the transformer: embedding size, MLP hidden layer size, number of heads, head size, and number of layers. Each factor of 2 increase in the number of parameters scales either one or two of these dimensions.

## Appendix G Qualitative Analysis Results

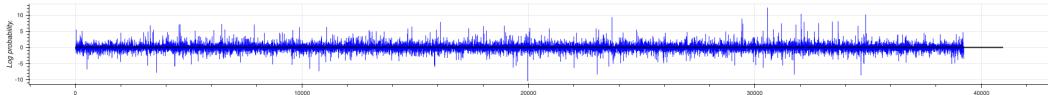


Figure 4: Difference in per-token cross-entropy loss.

The following are excerpts from our qualitative study. We selected five books at random from PG19 test set, and ran two different models on each book. The first model is a 13-layer Transformer-XL, and the second is the Rec:fixed:skip configuration of the Block-Recurrent Transformer. For each token, we compute the difference between the cross-entropy loss (i.e., the negative log likelihood (NLL)) output by both models, and then sort the results.

Figure 4 shows an example of the per-token difference in NLL between the two models on the first book; the x-axis is the index of the token. On average, the recurrent model does slightly better than Transformer-XL, but it does not necessarily make a better prediction for any individual token.

The following excerpts show the top 4 tokens where the Block-Recurrent Transformer made a better prediction than Transformer-XL; these tokens correspond to spikes in Figure 4. We show the token number, the NLL returned by the recurrent model, the NLL returned by Transformer-XL, and an excerpt of text, with the token itself marked with |token|. Almost all of the top tokens are proper names of characters and places. In all cases except one, the mispredicted name does not appear within the attention window of the previous 512 tokens. These names are thus invisible to Transformer-XL, but visible to the recurrent model.

Note that these are not cherry picked examples; the five books are chosen at random. Moreover, the same pattern still holds if the search is expanded to the top 40 tokens for each book. In fact, even the names are often the same; Transformer-XL often seems to mispredict the same names over and over again; these are likely the names of main characters.

Sorting the other way, to show the top tokens where Transformer-XL does better than the recurrent model, does not show the same pattern. There are still plenty of proper names, but it is usually cases where both models fail to predict the name. Moreover, the names are mixed with more common words as well.

*Memoirs of Marie Antoinette Queen Of France Vol. 7.*

(30555, 0.3696011, 12.688916)

physician's house to make inquiries as to the cause of so long an absence. Gomin and Larne had not yet ventured to follow this advice, when next

(32043, 0.20177796, 10.513703)

with the autopsy arrived at the outer gate of the Temple. These were Dumangin, head physician of the Hospice de l'Unite; Pelletan,

(34896, 1.3752508, 11.534926)

who had acted as courier to Louis XVI. during the flight to Varennes, and Turgil, who had waited on the Princesses in the Temple. It was

(34896, 1.3752508, 11.534926)

. In all the evidence there appeared but two serious facts, attested by Latour-du-Pin and Valaze, who deposed to them because they could

Notes: "Gomin" appears multiple times in the top 40 results.

### *An Unsentimental Journey through Cornwall*

(983, 0.80441666, 11.626528)

OUGH CORNWALL [Illustration: FALMOUTH, FROM FLUSHING.]  
|DAY| THE FIRST I believe in holidays. Not in a frantic rushing

(23606, 0.655162, 11.265186)

there was not the slightest use in getting up, I turned round and took another sleep. |DAY| THE FIFTH "Hope for the best, and be

(11297, 0.405902, 10.426135)

ball. "Ma'am, if you go slow and steady, with me before and Curgenven behind, you'll not fall." Nor did I. I record it

(38021, 1.3457009, 11.167879)

our journey; going over the same ground which we had traversed already, and finding Pradenack Down as bleak and beautiful as ever. Our first

Notes: The word "DAY" appears earlier in the table of contents, but not within the previous 512 tokens. "Curgenven" appears multiple times in the top 40.

### *The Good Hope by Herman Heijermans Jr.*

(3975, 0.7425603, 15.8969145)

to us." It matters nothing that this gospel of Life has often been preached. Heijermans has caught the spirit of it as well as the letter.

(7385, 0.26241615, 15.000762)

must scratch the stones. CLEM. Tomorrow afternoon, then. COB. Tja! I'll be here, then. Good day, Miss. [To Barend.] Good

(42638, 0.73628587, 15.407666)

hundred guilders. Bejour. [Rings off; at the last words Kneirtje has entered.] KNEIRTJE. [Absentely.] I---[

(25798, 0.12310324, 14.402218)

They exeunt, dragging Barend.] KNEIR. Oh, oh---- TRUUS!. [With anxious curiosity, at side door.] What was the matter, Kneir?

Notes: The word "Tja" is not a proper name, but is a Dutch word that is likely unique to this particular book. It appears multiple times in the top 40 results. The name "Truu" appears multiple times in the top 40.

### *Travels in Morocco Volume 2 by James Richardson*

Notes: The second example in this list is not a good one, because both models mispredict the name. We thus include a fifth example as well. The word “Toser” appears frequently in the top 40, and is also the only case where the proper name appears within the attention window of 512 tokens.

(61049, 2.0328524, 18.696453)

, some of them as black as [Redacted]s. Many people in T|oser| have sore eyes, and several with the loss of one eye, or nearly so; opthal

(63453, 9.254061, 24.413633)

Tunis;" but the restrictive system established by the Turks during late years at Ghad|umes|, has greatly damaged the trade between the Jereed and

(66149, 0.8768588, 14.105822)

enterprizing fellow, worthy of imitation. He calculated the distance from Ghabs to T|oser| at 200 miles. There are a number of towns in the

(64161, 1.0309317, 14.152167)

and ourselves went to Wedyen, a town and date-wood about eight miles from T|oser|, to the left. The date-grove is extensive, and there are

(27282, 0.04982663, 12.551973)

, is a very ancient city, situate upon the right bank of the river Boura|gralg, and near its mouth. This place was captured in 1263, by

### *India’s Love Lyrics by Laurence Hope*

(15694, 0.07477246, 11.22447)

rieving A wasted chance. Fate knows no tears. Verses:  
Fa|iz|Ulla Just in the hush before dawn A little wistful wind is

(3135, 0.0126608405, 9.759871)

afloat In the little noon tide of love’s delights Between  
two Nights. Val|go|vind’s Boat Song Waters glisten and  
sunbeams quiver,

(23183, 0.1419289, 9.481476)

sleep, the touch of your lips on my mouth. His Rubies: Told by  
Val|go|vind Along the hot and endless road, Calm and erect,  
with haggard eyes,’

(26886, 0.13428347, 9.46193)

off sheath, And find there is no Rival like the Past. Verse  
by T|aj|Mahomed When first I loved, I gave my very soul Utterly

## G.1 Clearing the recurrent state

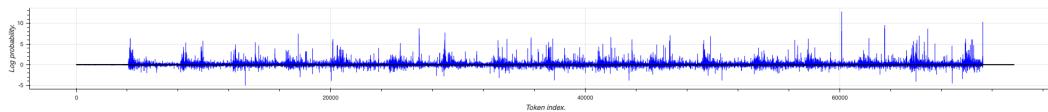


Figure 5: Difference in per-token cross-entropy loss with state clearing.

Our second qualitative study is structured similarly to the first, except that instead of comparing two different models, we compare two different runs of the same model: the Rec:fixed:skip configuration. The first run processes the book normally, while the second run clears the recurrent states at the beginning of each 4096-token segment. In the second run, the model can use recurrence within a segment to look beyond the local attention window of 512 tokens, but it cannot use recurrence to carry information from one segment to the next.

This experiment is somewhat cleaner than the first, because both runs are done with the same pre-trained model, which has the same parameters. Figure 5 shows the difference in per-token cross-entropy loss for the first book. There is no difference between the two runs for the first segment, and the biggest differences in subsequent segments are often clustered near the start of each new segment.

The overall pattern is very similar to the first qualitative experiment: most of the tokens involve proper names. We verified that in most cases, the mispredicted name not only does not occur within the 512-token attention window, but does not occur within the 4096-token segment. In addition to proper names, chapter titles and illustration captions occur frequently within the top 40 results; the recurrent model seems to be remembering these from a previous occurrence in the table of contents.

**Perhaps most interestingly, in two of the books, one of the highest ranked mispredictions was the title and author of the book itself.** The Gutenberg project inserts boilerplate at both the beginning and end of each book; the title and author are listed multiple times at the beginning, and once at the end. This experiment thus shows that the model is able to “remember” this information in the recurrent state, across a distance of 60,000 tokens or more.

### ***Baby Mine, by Margaret Mayo***

Note that the 2nd misprediction **is the title of the book itself**, at token number 71,244.

(60153, 0.00438364, 12.798895)

nerves, but you needn't worry, I've got everything fixed.  
Donneg|hey| sent a special officer over with me. He's outside

(71244, 2.3727102, 12.660636)

sunlight and shadows of his ample, well kept grounds. End of the  
Project Gutenberg EBook of Baby| Mine, by Margaret Mayo \*\*\*

(63536, 4.132567, 13.688847)

Alfred, with the air of a connoisseur. "She sure is," admitted  
Don|neg|hey, more and more disgruntled as he felt his reputation for

(26947, 3.7521973, 12.516711)

with a sigh of thanksgiving he hurried upstairs to his unanswered mail.  
CHAPTER XIII When Alfred| Hardy| found himself on the train bound for

### ***The 'Patriotes' of '37 by Alfred D. Decelles***

(38759, 0.85374373, 11.140007)

24, 125, 126. Cote, Dr Cyri|le|, 89, 108, 118, 120;

(40181, 0.35475963, 10.291676)

17-26, 129-30. Nelson, Dr Wolf|red|, a follower of Papineau, 37, 60

(28756, 0.0010796335, 9.256147)

on a well-reasoned plan of action. Most of the leaders--Wolf|red| Nelson,  
Thomas Storrow Brown, Robert Bouchette, and Amury Girod--were

(28772, 1.7782648, 10.611834)

leaders--Wolfred Nelson, Thomas Storrow Brown, Robert Bouchette, and  
Am|ury| Girod--were strangers to the men under their command; and none of

### ***Another Brownie Book, by Palmer Cox***

In this case, **the top 4 results include the author of the book**. This book has a lot of illustrations which are listed in the table of contents, and make up many of the other results in the top 40.

(16442, 0.087840006, 8.706101)

[Illustration] [Illustration] [Illustration] THE BROWNIES  
AND THE TUG|BO|AT. While Brownies strayed along a  
(24225, 0.04784866, 8.266858)  
[Illustration] End of the Project Gutenberg EBook of Another  
Brownie Book, by Palmer| Cox| \*\*\*  
(24224, 5.651471, 11.840027)  
Illustration] [Illustration] End of the Project Gutenberg  
EBook of Another Brownie Book, by| Palmer| Cox \*\*\*  
(4460, 1.0682098, 6.6397715)  
To secret haunts without delay. [Illustration] [Illustration]  
THE BROWNIES AT| AR|CHERY. [Illustration] [Illustration]

### *The Life Of Thomas Paine Vol. II. (of II) by Moncure Daniel*

By the end of the book, the recurrent model is quite sure that "Paine" is a main character, but not if its memory keeps getting erased.

(170457, 0.98941976, 12.209277)  
they could. The scandal branched into variants. Twenty-five years later  
pious Grant| Thor|burn promulgated that Paine had run off from Paris  
(120592, 0.6974209, 10.396527)  
my friends and accept the same to yourself." As the Commissioners did  
not leave when they expected,| Paine| added several other letters to  
(169152, 0.18432029, 9.516743)  
whose composition the farrier no doubt supposed he had paid the editor  
with stories borrowed from "Old|ys|," or not actionable. Cheetham  
(139870, 0.53406477, 9.646917)  
nor was any letter received from him. This was probably the most important  
allusion in a letter of| Paine|, dated New York, March 1, 1804, to "C

## Appendix H Token level cross-entropy

In addition to qualitative studies comparing a single document, we also compared the average bits-per-token over all documents in the PG19 test set. It has been observed that in vanilla transformer architectures, this token-wise cross-entropy often diverges at long segment lengths [53].

In Figure 6 we plot the cumulative cross-entropy, which is the average bits-per-token (i.e.  $\log_2$  perplexity) averaged up to the given length, and we compare the Block Recurrent Transformer against the Transformer-XL baseline. Performance of the two architectures is comparable for the first few thousand tokens, but the recurrent architecture clearly outperforms Transformer-XL at longer document lengths.

## Appendix I Vocabulary Ablation Experiments

The authors of the PG-19 dataset propose *word-level perplexity* (WLP) as a measure of model performance [15]. However, it is only possible to compute WLP if there is an accurate count of the number of words. A published word count is provided for the PG19 data set [15], but a word count is not available for the other data sets in this paper, such as arXiv and github. There is not even a clear definition of what constitutes a “word” in source code or L<sup>A</sup>T<sub>E</sub>X documents, which have a lot of symbols. Consequently, we do not provide WLP for these data sets.

WLP also glosses over some details of tokenization; in particular, the word count ignores whitespace. SentencePiece [44] is often advertised as being a subword tokenizer that preserves whitespace, but in

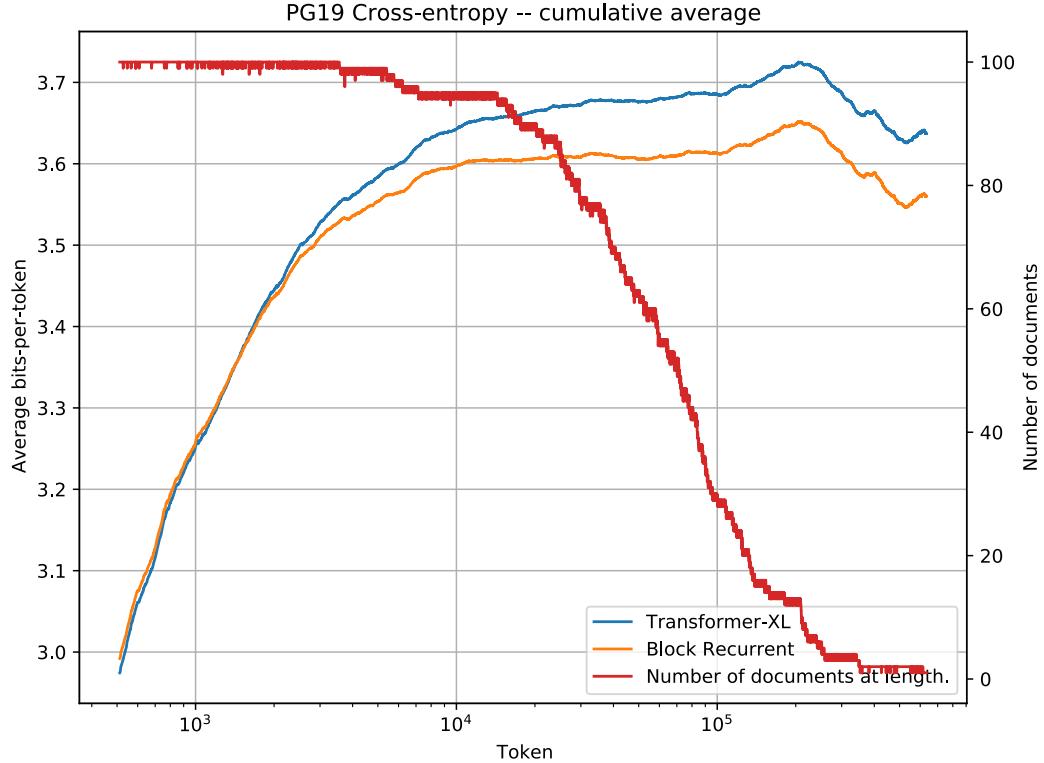


Figure 6: Cumulative cross-entropy on PG19 of a 13-layer Transforemer-XL and Block Recurrent model. Though comparable at the first few thousand tokens, the recurrent model performs better at longer sequences. In red we show the number of documents at a given token length.

fact that is only partially true. Depending on how it is configured, SentencePiece may normalize text by mapping multiple whitespace characters (e.g. tab) to space, merge internal whitespace characters, or strip all newlines from the text.

Thus, depending on vocabulary, a model trained with a SentencePiece tokenizer may or may not be predicting whitespace and newlines. If it does predict them, then that whitespace is not captured by the word count, which can affect the WLP numbers.

Measuring bits-per-character (BPC) rather than WLP is a potentially simpler and fairer comparison that works with any data set and tokenizer, but it runs into the same problem with whitespace. If the tokenizer merges or strips whitespace from the input text, as many SentencePiece vocabularies do, then it is not possible to get a consistent character count across different vocabularies.

### I.1 Effect of vocabulary on model perplexity

The vocabulary itself can also make a significant difference. We argue that the tokenizer and vocabulary should be considered as part of the model, and comparisons using either WLP or BPC are only valid between models that use the same tokenizer and vocabulary. Using WLP to judge model innovations without taking into account other differences can lead to incorrect conclusions. We give an empirical example of how using different vocabularies can lead to significant differences in WLP.

The total cross-entropy loss  $L_S = - \sum_{t=0}^{N_S} \log(p_t | p_{<t})$  where  $S$  is the chosen tokenization scheme and  $N_S$  is the number of tokens in the data when tokenized by  $S$ . Models using the same tokenization scheme can be compared by their *token-level perplexity*  $\text{PPL}_S = \exp(\frac{L_S}{N_S})$  where  $\frac{L_S}{N_S}$  is the *average loss*. Lower perplexity is better.

WLP is defined to be  $\exp\left(\frac{L_S}{N_W}\right)$  where  $N_W$  is the number of "words" as defined by the authors of the PG-19 datasets. For the PG-19 test set,  $N_W = 6,966,499$  [15]. Thus, WLP is simply a rescaling of the average loss, i.e.  $\text{WLP} = \exp\left(\frac{L_S}{N_S} \frac{N_S}{N_W}\right)$ , where we'll refer to  $\frac{N_S}{N_W}$  as the *scale factor* – the number of tokens per word.

### I.1.1 Vocabulary size

The size of the vocabulary has three obvious effects on the model. First, a larger vocabulary has more parameters, and thus can store more information in the embedding table.

Second, a larger vocabulary will typically have longer and rarer words, so the average length of each token (characters-per-token) will also be longer, and the *scale factor* (tokens-per-word) will be smaller. The context length of the model *in tokens* is usually fixed, so having longer tokens means that the model can attend over longer distances in the input text.

Third, having longer tokens also means that the model will process more of the training data on each training step (again because the number of tokens per step is fixed), and thus complete more epochs over the data set.

### I.1.2 Vocabulary quality

Even if two vocabularies are the same size, there may be a difference in vocabulary quality. For example, if a multi-lingual vocabulary is used on English-language text, then only a portion of its total capacity will be used to capture English words, and it will behave much like a smaller vocabulary.

A more subtle issue is that there are many ways in which text can be tokenized. A tokenizer which does a good job of capturing natural semantic segmentation into subwords can be expected to perform better than one that doesn't, e.g. "walk|ed" vs. "wal|ked."

## I.2 Experimental results

We train the `Rec:fixed:skip` model with a segment length of 4096 for 500k steps employing an inverse square root decay schedule while only varying the choice of vocabulary (all other hyperparameters are as in Section 4).

The byte-level vocabulary does not use a tokenizer, and operates on the raw ASCII text. The T5 vocabulary [39] was trained on multi-lingual data from Common Crawl (commoncrawl.org), consists of 32k tokens, and does not preserve whitespace. We also test on a 32k sized version of the LaMDA vocabulary [54]. To test the effect of vocabulary size, we also train a number of SentencePiece models on the PG19 training set with sizes ranging from 512 to 128k.

Results are shown in Table 5 and Figure 7. For SentencePiece vocabularies trained on PG19, we observe that a larger vocabulary has a significant effect on the scale factor, although the effect starts to diminish with more than 32k entries (see Figure 7). The average bits per token increases as the tokens become longer.

# GENERALIZATION THROUGH MEMORIZATION: NEAREST NEIGHBOR LANGUAGE MODELS

Urvashi Khandelwal<sup>†\*</sup>, Omer Levy<sup>‡</sup>, Dan Jurafsky<sup>†</sup>, Luke Zettlemoyer<sup>‡</sup> & Mike Lewis<sup>‡</sup>

<sup>†</sup>Stanford University

<sup>‡</sup>Facebook AI Research

{urvashik, jurafsky}@stanford.edu

{omerlevy, lsz, mikelewis}@fb.com

## ABSTRACT

We introduce  $k$ NN-LMs, which extend a pre-trained neural language model (LM) by linearly interpolating it with a  $k$ -nearest neighbors ( $k$ NN) model. The nearest neighbors are computed according to distance in the pre-trained LM embedding space, and can be drawn from any text collection, including the original LM training data. Applying this augmentation to a strong WIKITEXT-103 LM, with neighbors drawn from the original training set, our  $k$ NN-LM achieves a new state-of-the-art perplexity of 15.79 – a 2.9 point improvement with no additional training. We also show that this approach has implications for efficiently scaling up to larger training sets and allows for effective domain adaptation, by simply varying the nearest neighbor datastore, again without further training. Qualitatively, the model is particularly helpful in predicting rare patterns, such as factual knowledge. Together, these results strongly suggest that learning similarity between sequences of text is easier than predicting the next word, and that nearest neighbor search is an effective approach for language modeling in the long tail.

## 1 INTRODUCTION

Neural language models (LMs) typically solve two subproblems: (1) mapping sentence prefixes to fixed-sized representations, and (2) using these representations to predict the next word in the text (Bengio et al., 2003; Mikolov et al., 2010). We present a new language modeling approach that is based on the hypothesis that the representation learning problem may be easier than the prediction problem. For example, any English speaker knows that *Dickens is the author of* and *Dickens wrote* will have essentially the same distribution over the next word, even if they do not know what that distribution is. We provide strong evidence that existing language models, similarly, are much better at the first problem, by using their prefix embeddings in a simple nearest neighbor scheme that significantly improves overall performance.

We introduce  $k$ NN-LM, an approach that extends a pre-trained LM by linearly interpolating its next word distribution with a  $k$ -nearest neighbors ( $k$ NN) model. The nearest neighbors are computed according to distance in the pre-trained embedding space and can be drawn from any text collection, including the original LM training data. This approach allows rare patterns to be memorized explicitly, rather than implicitly in model parameters. It also improves performance when the same training data is used for learning the prefix representations and the  $k$ NN model, strongly suggesting that the prediction problem is more challenging than previously appreciated.

To better measure these effects, we conduct an extensive empirical evaluation. Applying our  $k$ NN augmentation to a strong WIKITEXT-103 LM using only the original dataset achieves a new state-of-the-art perplexity of 15.79 – a 2.86 point improvement over the base model (Baevski & Auli, 2019) – with no additional training. We also show that the approach has implications for efficiently scaling up to larger training sets and allows for effective domain adaptation, by simply varying the nearest neighbor datastore. Training a model on 100-million tokens and using  $k$ NN search over a 3-billion token dataset can outperform training the same model on all 3-billion tokens, opening a

\*Work done while the first author was interning at Facebook AI Research.

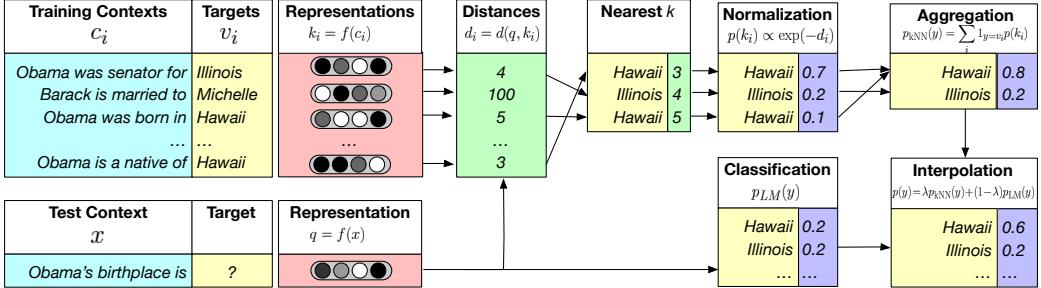


Figure 1: An illustration of  $k$ NN-LM. A datastore is constructed with an entry for each training set token, and an encoding of its leftward context. For inference, a test context is encoded, and the  $k$  most similar training contexts are retrieved from the datastore, along with the corresponding targets. A distribution over targets is computed based on the distance of the corresponding context from the test context. This distribution is then interpolated with the original model’s output distribution.

new path for efficiently using large datasets in language models. Similarly, adding out-of-domain data to the datastore makes a single LM useful across multiple domains, again without further training. Qualitatively, we find the model is particularly helpful for long-tail patterns, such as factual knowledge, which might be easier to access via explicit memory.

## 2 NEAREST NEIGHBOR LANGUAGE MODELING

Language models (LMs) assign probabilities to sequences. Given a *context* sequence of tokens  $c_t = (w_1, \dots, w_{t-1})$ , autoregressive LMs estimate  $p(w_t|c_t)$ , the distribution over the *target* token  $w_t$ .

The  $k$ NN-LM involves augmenting such a pre-trained LM with a nearest neighbors retrieval mechanism, without any additional training (the representations learned by the LM remain unchanged). This can be done with a single forward pass over a text collection (potentially including the original LM training set), where the resulting context-target pairs are stored in a key-value datastore that is queried during inference, as illustrated in Figure 1.

**Datastore** Let  $f(\cdot)$  be the function that maps a context  $c$  to a fixed-length vector representation computed by the pre-trained LM. For instance, in a Transformer LM,  $f(c)$  could map  $c$  to an intermediate representation that is output by an arbitrary self-attention layer. Then, given the  $i$ -th training example  $(c_i, w_i) \in \mathcal{D}$ , we define the key-value pair  $(k_i, v_i)$ , where the key  $k_i$  is the vector representation of the context  $f(c_i)$  and the value  $v_i$  is the target word  $w_i$ . The datastore  $(\mathcal{K}, \mathcal{V})$  is thus the set of all key-value pairs constructed from all the training examples in  $\mathcal{D}$ :

$$(\mathcal{K}, \mathcal{V}) = \{(f(c_i), w_i) | (c_i, w_i) \in \mathcal{D}\} \quad (1)$$

**Inference** At test time, given the input context  $x$  the model generates the output distribution over next words  $p_{LM}(y|x)$  and the context representation  $f(x)$ . The model queries the datastore with  $f(x)$  to retrieve its  $k$ -nearest neighbors  $\mathcal{N}$  according to a distance function  $d(\cdot, \cdot)$  (squared  $L^2$  distance in our experiments, making the similarity function an RBF kernel). Then, it computes a distribution over neighbors based on a softmax of their negative distances, while aggregating probability mass for each vocabulary item across all its occurrences in the retrieved targets (items that do not appear in the retrieved targets have zero probability):

$$p_{kNN}(y|x) \propto \sum_{(k_i, v_i) \in \mathcal{N}} \mathbb{1}_{y=v_i} \exp(-d(k_i, f(x))) \quad (2)$$

Finally, we follow [Grave et al. \(2017a\)](#) and interpolate the nearest neighbor distribution  $p_{kNN}$  with the model distribution  $p_{LM}$  using a tuned parameter  $\lambda$  to produce the final  $k$ NN-LM distribution:

$$p(y|x) = \lambda p_{kNN}(y|x) + (1 - \lambda) p_{LM}(y|x) \quad (3)$$

**Implementation** The datastore contains an entry for each target in the training set, which for LMs can be up to billions of examples. To search over this large datastore, we use FAISS (Johnson et al., 2017), an open source library for fast nearest neighbor retrieval in high dimensional spaces. FAISS speeds up search by clustering the keys and looking up neighbors based on the cluster centroids, while reducing memory usage by storing compressed versions of the vectors. We found in preliminary experiments that using  $L^2$  distance for FAISS retrieval results in better performance for  $k$ NN-LM, compared to inner product distance.

**Related Cache Models** Prior work (Grave et al., 2017c; Merity et al., 2017) used a similar approach to compute similarity to the previous hidden states of *test* documents, making it easier to copy rare vocabulary items from the recent past. Such techniques have been less popular since the development of Transformers (Vaswani et al., 2017), which can learn to copy recent words using self-attention; in Section 4.1 we observe relatively small gains from caching recent items in the same test document à la Grave et al. (2017c). Most relatedly, Grave et al. (2017a) describe an *online* language model using nearest neighbor search over all previous hidden states, to improve domain adaptation. In our work, we only save training data, with the goal of explicitly memorizing training examples to better generalize to similar cases at test time.

### 3 EXPERIMENTAL SETUP

**Data** Experiments in this paper use the following English corpora:

WIKITEXT-103 is a standard benchmark by Merity et al. (2017) for autoregressive language modeling with a 250K word-level vocabulary. It consists of 103M tokens of Wikipedia in the training set and 250K tokens in each of the development and test sets.

BOOKS is the Toronto Books Corpus (Zhu et al., 2015), containing 0.7B. Complete books are held out for validation/test.

WIKI-3B is English Wikipedia, containing about 2.87B tokens. Whole articles are held out for validation/test.

WIKI-100M is a random 100M token subset of WIKI-3B, consisting of complete articles.

Except for WIKITEXT-103, text is tokenized using the byte-pair encoding (Sennrich et al., 2015) with the 29K subword vocabulary from BERT (Devlin et al., 2019).

**Model Architecture**  $k$ NN-LM is compatible with any model that produces fixed size context representations. We use decoder-only Transformers (Vaswani et al., 2017) for language modeling, which are the current state of the art. Since the  $k$ NN-LM makes no changes to the underlying LM, we take the exact architecture and optimization described by Baevski & Auli (2019) and use it to create a  $k$ NN-LM for inference. This model consists of 16 layers, each with 16 self-attention heads, 1024 dimensional hidden states, and 4096 dimensional feedforward layers, amounting to 247M trainable parameters. It processes 3072 tokens of context per example for WIKITEXT-103 and 1024 tokens for the rest of the corpora. Following Baevski & Auli (2019), we use adaptive inputs and an adaptive softmax (Grave et al., 2017b) with tied weights (Press & Wolf, 2017) for the WIKITEXT-103 experiments. On other datasets we do not use adaptive inputs or an adaptive softmax.

**Evaluation** LMs are trained to minimize the negative log-likelihood of the training corpus, and evaluated by perplexity (exponentiated negative log-likelihood) on held out data. Following Baevski & Auli (2019), 512 tokens are scored per test example, but up to 2560 tokens of extra prior context is provided for WIKITEXT-103 and up to 512 tokens of extra prior context is provided for the rest of the corpora.

**$k$ NN-LM** The keys used for  $k$ NN-LM are the 1024-dimensional representations fed to the feed-forward network in the final layer of the Transformer LM (after self-attention and layernorm; see Section 5 for further explanation). We perform a single forward pass over the training set with the trained model, in order to save the keys and values. During this forward pass, each target token is provided a minimum of 1536 tokens of prior context for WIKITEXT-103 and a minimum of 512

Model	Perplexity ( $\downarrow$ )		# Trainable Params
	Dev	Test	
Baevski & Auli (2019)	17.96	18.65	247M
+Transformer-XL (Dai et al., 2019)	-	18.30	257M
+Phrase Induction (Luo et al., 2019)	-	17.40	257M
Base LM (Baevski & Auli, 2019)	17.96	18.65	247M
+ $k$ NN-LM	<b>16.06</b>	<b>16.12</b>	247M
+Continuous Cache (Grave et al., 2017c)	17.67	18.27	247M
+ $k$ NN-LM + Continuous Cache	<b>15.81</b>	<b>15.79</b>	247M

Table 1: Performance on WIKITEXT-103. The  $k$ NN-LM substantially outperforms existing work. Gains are additive with the related but orthogonal continuous cache, allowing us to improve the base model by almost 3 perplexity points with no additional training. We report the median of three random seeds.

Model	Perplexity ( $\downarrow$ )		# Trainable Params
	Dev	Test	
Base LM (Baevski & Auli, 2019)	14.75	11.89	247M
+ $k$ NN-LM	<b>14.20</b>	<b>10.89</b>	247M

Table 2: Performance on BOOKS, showing that  $k$ NN-LM works well in multiple domains.

tokens for the rest of the corpora. A FAISS index is then created using 1M randomly sampled keys to learn 4096 cluster centroids. For efficiency, keys are quantized to 64-bytes. During inference, we retrieve  $k = 1024$  neighbors, and the index looks up 32 cluster centroids while searching for the nearest neighbors. For WIKITEXT-103 experiments, we compute squared  $L^2$  distances with full precision keys, but for the other datasets we use the FAISS  $L^2$  distances (not squared) between quantized keys directly, for faster evaluation. We tune the interpolation parameter  $\lambda$  on the validation set.<sup>1</sup>

**Computational Cost** Although the  $k$ NN-LM requires no training given an existing LM, it does add some other computational overheads. Storing the keys and values requires a single forward pass over the training set, which amounts to a fraction of the cost of training for one epoch on the same examples. Once the keys are saved, for WIKITEXT-103 building the cache with 103M entries takes roughly two hours on a single CPU. Finally, running on the validation set took approximately 25 minutes when retrieving 1024 keys. While the cost of building a large cache grows linearly in the number of entries, it is trivial to parallelize and requires no GPU-based training.

## 4 EXPERIMENTS

### 4.1 USING THE TRAINING DATA AS THE DATASTORE

We first experiment with creating a datastore from the same data used to train the LM. Table 1 shows that  $k$ NN-LM improves perplexity on WIKITEXT-103 from 18.65 (Baevski & Auli, 2019) to a new state-of-the-art of 16.12. We also provide reported perplexities from two other recent models that also build upon Baevski and Auli’s, suggesting that further improvements may be possible by augmenting the  $k$ NN-LM with these techniques. We compare with models trained only on the standard training set, but recent work has shown performance can be improved by training on additional data, from either the test set (Krause et al., 2019) or large amounts of web text (Shoeybi et al., 2019).

We also experiment with a continuous cache model, a related but orthogonal technique from Grave et al. (2017c), in which the model saves and retrieves neighbors from earlier in the test document,

<sup>1</sup>Code is available at: <https://github.com/urvashik/knnlm>

Training Data	Datastore	Perplexity ( $\downarrow$ )	
		Dev	Test
WIKI-3B	-	16.11	15.17
WIKI-100M	-	20.99	19.59
WIKI-100M	WIKI-3B	14.61	13.73

Table 3: Experimental results on WIKI-3B. The model trained on 100M tokens is augmented with a datastore that contains about 3B training examples, outperforming the vanilla LM trained on the entire WIKI-3B training set.

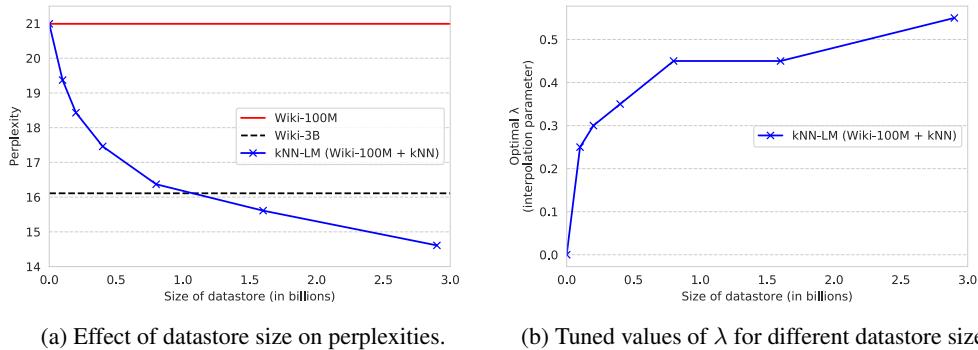


Figure 2: Varying the size of the datastore. (a) Increasing the datastore size monotonically improves performance, and has not saturated even at about 3B tokens. A kNN-LM trained on 100M tokens with a datastore of 1.6B tokens already outperforms the LM trained on all 3B tokens. (b) The optimal value of  $\lambda$  increases with the size of the datastore.

rather than the training set. Gains from interpolating with the continuous cache are smaller than reported in the original setting that used LSTMs, perhaps because self-attentive language models can learn to perform such queries. Improvements from the continuous cache are additive with the kNN-LM, pushing our state-of-the-art result to 15.79, a gain of 2.86 over the base model.

Finally, we repeat the experiment using text from a different domain, BOOKS, to control for the possibility that encyclopedic Wikipedia text is somehow uniquely good for caching. Table 2 shows an improvement in test set perplexity from 11.89 to 10.89, suggesting that this is not the case.

#### 4.2 MORE DATA WITHOUT TRAINING

Section 4.1 has shown that retrieving neighbors from the training data can significantly improve language modeling performance. This raises the question: can retrieving nearest neighbors from data be a substitute for training on it? To test this, we train a LM on WIKI-100M and use it to build a datastore from WIKI-3B, a corpus 30 times larger than the training set. We then compare this kNN-LM to a vanilla LM trained on the entire WIKI-3B corpus.<sup>2</sup>

Table 3 shows that, as expected, the model trained on 3B tokens dramatically outperforms the model trained on 100M tokens, improving perplexity from 19.59 to 15.17. However, adding nearest neighbors retrieval over those 3B examples to the model trained on 100M tokens improves perplexity from 19.59 to 13.73; i.e. *retrieving nearest neighbors from the corpus outperforms training on it*. This result suggests that rather than training language models on ever larger datasets, we can use smaller datasets to learn representations and augment them with kNN-LM over a large corpus.

<sup>2</sup>The original LM (Baevski & Auli, 2019) was trained for 286K steps on a corpus of similar size to WIKI-100M. When scaling up to WIKI-3B, we tuned only the number of updates on the validation set and found that training for 572K steps (double) produces a slightly stronger baseline.

Training Data	Datastore	Perplexity ( $\downarrow$ )	
		Dev	Test
WIKI-3B	-	37.13	34.84
BOOKS	-	14.75	11.89
WIKI-3B	BOOKS	24.85	20.47

Table 4: Domain adaptation experiments, with results on BOOKS. Adding an in-domain datastore to a Wikipedia-trained model improves results by 23 points, approaching in-domain training.

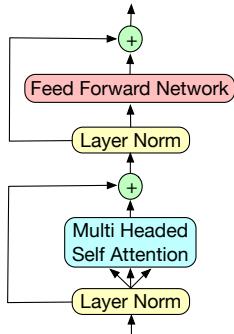


Figure 3: Transformer LM layer.

Key Type	Dev ppl. ( $\downarrow$ )
No datastore	17.96
Model output	17.07
Model output layer normalized	17.01
FFN input after layer norm	<b>16.06</b>
FFN input before layer norm	17.06
MHSA input after layer norm	16.76
MHSA input before layer norm	17.14

Table 5: WIKITEXT-103 validation results using different states from the final layer of the LM as the representation function  $f(\cdot)$  for keys and queries. We retrieve  $k=1024$  neighbors and  $\lambda$  is tuned for each.

To understand how the amount of data used for  $k$ NN retrieval affects performance, we use the WIKI-100M model to create datastores using different amounts of randomly sampled data from WIKI-3B. Figure 2a shows that using only 1.6B examples for the datastore already surpasses the performance of the model trained on all of WIKI-3B. In addition, performance does not saturate at 3B examples in the datastore, suggesting that growing the datastore more could lead to further gains. Figure 2b shows the model relies more on the  $k$ NN component as the size of the datastore increases.

#### 4.3 DOMAIN ADAPTATION

We also experiment with domain adaptation by creating a datastore on the target domain training set. Table 4 shows that an in-domain LM on BOOKS has a relatively low perplexity (11.89), while a model trained on WIKI-3B performs poorly on the BOOKS domain (34.84 perplexity). Adding  $k$ NN search over BOOKS to the WIKI-3B model reduces perplexity by 14 points (to 20.47), demonstrating that  $k$ NN-LM allows a single model to be useful in multiple domains, by simply adding a datastore per domain.

### 5 TUNING NEAREST NEIGHBOR SEARCH

While the  $k$ NN-LM is conceptually straightforward, and requires no additional training, a number of hyperparameters are introduced for nearest neighbor search. We experiment with different choices here.

**Key Function** For similarity search, we extract a representation of context  $c$  using an intermediate state of the LM  $f(c)$ . Transformers compute a number of different intermediate states, and we compare several choices depicted in Figure 3, with results shown in Table 5. While all the instantiations of  $f$  we tried are helpful, we achieved the largest improvement by using the input to the final layer’s feedforward network. We also observe that normalized representations (i.e. taken immediately after the layer norm) perform better. Repeating the experiment on the second-last transformer layer showed similar trends with slightly worse results (not shown), suggesting that the feedforward layer might be focusing more on the prediction problem, while the onus of representing the input falls more on the self-attention layer.

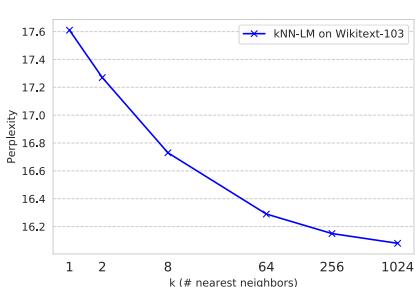


Figure 4: Effect of the number of nearest neighbors returned per word on WIKITEXT-103 (validation set). Returning more entries from the datastore monotonically improves performance.

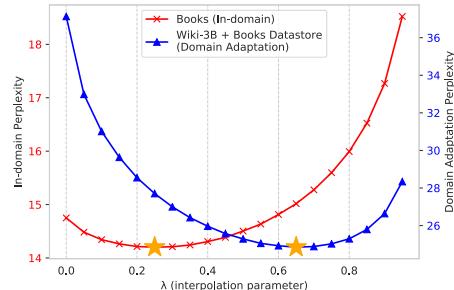


Figure 5: Effect of interpolation parameter  $\lambda$  on in-domain (left y-axis) and out-of-domain (right y-axis) validation set performances. More weight on  $p_{kNN}$  improves domain adaptation.

**Number of Neighbors per Query** Each query returns the top- $k$  neighbors. Figure 4 shows that performance monotonically improves as more neighbors are returned, and suggests that even larger improvements may be possible with a higher value of  $k$ . Nonetheless, even a small number of neighbors ( $k = 8$ ) is enough to achieve a new state of the art.

**Interpolation Parameter** We use a parameter  $\lambda$  to interpolate between the base model distribution and the distribution from  $k$ NN search over the dataset. Figure 5 shows that  $\lambda = 0.25$  is optimal on WIKITEXT-103. However,  $\lambda = 0.65$  works best for domain adaptation results (Figure 5).

**Precision of Similarity Function** In FAISS, the nearest neighbor search computes  $L^2$  distances against quantized keys. We found results were improved from 16.5 perplexity on WIKITEXT-103 to 16.06 by computing squared  $L^2$  distances with full precision keys for Equation 2.

## 6 ANALYSIS

**Qualitative Analysis** To understand why  $k$ NN-LM improves performance, we manually examine cases in which  $p_{kNN}$  was significantly better than  $p_{LM}$ . Table 6 shows one such example, along with several others in Appendix A. The example shows an interesting case where the model matches the trigram *impact on the* in several retrieved neighbors, but puts almost all weight on the most relevant neighbor, thus adding more value than an  $n$ -gram LM.

In general, we find that examples where  $k$ NN-LM is most helpful typically contain rare patterns. Examples include factual knowledge, names, and near-duplicate sentences from the training set. In these cases, assigning train and test instances similar representations (via  $f(\cdot)$ ) appears to be an easier problem than implicitly memorizing the next word in model parameters.

**Simple vs Neural Representation** We observe that many long-tail phenomena manifest as rare  $n$ -grams (e.g. names). Is it therefore possible to interpolate an  $n$ -gram model with a Transformer LM, as an alternative to our  $k$ NN approach? Figure 7 shows little improvement from using  $n$ -gram LMs – 0.2 perplexity points (similarly to Bakhtin et al. (2018)). This result highlights the need to use the learned representation function  $f(\cdot)$  to measure similarity between more varied contexts.

**Implicit vs Explicit Memory** If a neural representation function is crucial for  $k$ NN-LM, could implicitly memorizing the training dataset in the neural network parameters replace the explicit memory in the datastore? To test this, we train a Transformer LM with no dropout. Figure 8 shows that this model eventually reaches zero training loss, indicating that it can make perfect predictions for all examples in the training set; the model has memorized the dataset. Naturally, the memorizing LM overfits, i.e. the training loss drops to 0 while the best validation perplexity is much higher at 28.59. For comparison, the vanilla Transformer LM (with dropout) has a much higher training loss (shown in Figure 8), but also generalizes better with a validation perplexity of 17.96. This result shows that the Transformer has sufficient capacity to memorize the training set.

Test Context ( $p_{kNN} = 0.998, p_{LM} = 0.124$ )	Test Target	
<i>it was organised by New Zealand international player Joseph Warbrick, promoted by civil servant Thomas Eyton, and managed by James Scott, a publican. The Natives were the first New Zealand team to perform a haka, and also the first to wear all black. They played 107 rugby matches during the tour, as well as a small number of Victorian Rules football and association football matches in Australia. Having made a significant impact on the...</i>	development	
Training Set Context	Training Set Target	Context Probability
<i>As the captain and instigator of the 1888-89 Natives – the first New Zealand team to tour the British Isles – Warbrick had a lasting impact on the...</i>	development	0.998
<i>promoted to a new first grade competition which started in 1900. Glebe immediately made a big impact on the...</i>	district	0.00012
<i>centuries, few were as large as other players managed. However, others contend that his impact on the...</i>	game	0.000034
<i>Nearly every game in the main series has either an anime or manga adaptation, or both. The series has had a significant impact on the...</i>	development	0.00000092

Figure 6: Example where the  $k$ NN model has much higher confidence in the correct target than the LM. Although there are other training set examples with similar local  $n$ -gram matches, the nearest neighbour search is highly confident of specific and very relevant context.

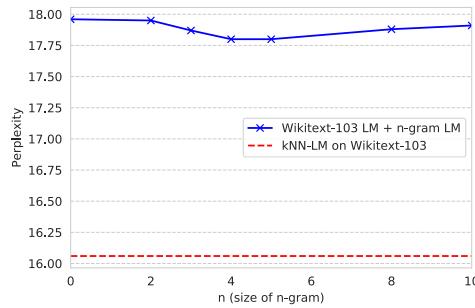


Figure 7: Interpolating the Transformer LM with  $n$ -gram LMs on WIKITEXT-103 (validation set). Using  $k$ NN-LM gives a much lower perplexity, suggesting that the representations are learning more than just matching local context.

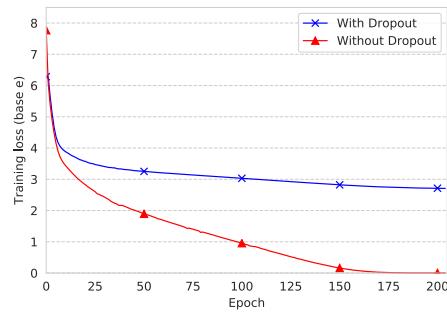


Figure 8: Training curves for the Transformer LM with and without dropout. Turning off dropout allows the training loss to go to 0, indicating that the model has sufficient capacity to memorize the training data.

We consider whether the memorizing LM can be an effective substitute for nearest neighbor search. Interpolating the memorizing LM with the original LM improves validation perplexity by just 0.1 – compared to 1.9 from  $k$ NN-LM. This result suggests that although the Transformer is expressive enough to memorize all training examples, learning to do so does not result in context representations that generalize. In contrast,  $k$ NN-LM memorizes training data while improving generalization.

From these experiments, we conjecture that  $k$ NN-LM improves performance because (1) the Transformer LM is very good at learning a representation function for contexts with an implicit notion of similarity, and (2) while the Transformer has capacity to memorize all training examples, doing so causes its representation to generalize less effectively, but (3) the  $k$ NN-LM allows the model to memorize the training data while retaining an effective similarity function.

## 7 RELATED WORK

We discuss related uses of caches for language modeling in Section 2.

Similar  $k$ NN models to ours have been proposed for computer vision tasks (Papernot & McDaniel, 2018; Orhan, 2018; Zhao & Cho, 2018), primarily motivated by improving interpretability and robustness to adversarial attacks. We hypothesize that our method may be particularly effective for language modeling, because plentiful unlabeled data allows datastores of billions of tokens, and language modeling often requires world knowledge to be learnt from few examples.

Nearest neighbor models have been applied to a number of NLP problems in the past, such as part of speech tagging (Daelemans et al., 1996) and morphological analysis (Bosch et al., 2007), but the use of learned representations makes the similarity function much more effective in the case of neural models. More recently, Kaiser et al. (2017) have used a similarly differentiable memory that is learned and updated during training, and is applied to one-shot learning tasks.

Several models have also improved language generation by using training examples directly at test time. Guu et al. (2018) propose a model that samples training sentences at random and edits them with a sequence-to-sequence model, but does not use a retrieval mechanism such as  $k$ NN. Gu et al. (2018) introduce a translation model that attends over retrieved training set examples. Weston et al. (2018) improve a dialogue response generation model by refining similar instances from the training set.  $k$ NN-LM differs from these approaches by working at the level of individual tokens instead of whole training sentences, as well as not incorporating the retrieval mechanism into the training pipeline.

A general trend in machine learning, and in language modeling in particular, is that adding more data consistently improves performance (Devlin et al., 2019; Radford et al., 2019; Yang et al., 2019; Liu et al., 2019; Zellers et al., 2019; Shoeybi et al., 2019). Our work offers an alternative method for scaling language models, in which relatively small models learn context representations, and a nearest neighbour search acts as a highly expressive classifier.

## 8 CONCLUSION AND FUTURE WORK

We have introduced  $k$ NN-LMs, which can significantly outperform standard language models by directly querying training examples at test time. The approach can be applied to any neural language model. The success of this method suggests that learning similarity functions between contexts may be an easier problem than predicting the next word from some given context. Future work should explore explicitly training similarity functions, and reducing the size of the datastore.

### ACKNOWLEDGMENTS

The authors thank the anonymous reviewers as well as Sida Wang, Kartikay Khandelwal, Kevin Clark and members of the FAIR Seattle team for helpful discussions and comments.

### REFERENCES

- Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In *ICLR*, 2019.
- Anton Bakhtin, Arthur Szlam, Marc’Aurelio Ranzato, and Edouard Grave. Lightweight adaptive mixture of neural and n-gram language models. *arXiv preprint arXiv:1804.07705*, 2018.
- Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.
- Antal van den Bosch, Bertjan Busser, Sander Canisius, and Walter Daelemans. An efficient memory-based morphosyntactic tagger and parser for dutch. *LOT Occasional Series*, 7:191–206, 2007.
- Walter Daelemans, Jakub Zavrel, Peter Berck, and Steven Gillis. Mbt: A memory-based part of speech tagger-generator. In *WVLC*, 1996.

## A APPENDIX

This section provides several examples where  $p_{kNN}$  places higher probability mass on the true target, compared to  $p_{LM}$ .

Test Context ( $p_{kNN} = 0.995, p_{LM} = 0.025$ )	Test Target	
<i>For Australians and New Zealanders the Gallipoli campaign came to symbolise an important milestone in the emergence of both nations as independent actors on the world stage and the development of a sense of national identity. Today, the date of the initial landings, 25 April, is known as Anzac Day in Australia and New Zealand and every year thousands of people gather at memorials in both nations, as well as Turkey, to...</i>	honour	
Training Set Context	Training Set Target	Context Probability
<i>Despite this, for Australians and New Zealanders the Gallipoli campaign has come to symbolise an important milestone in the emergence of both nations as independent actors on the world stage and the development of a sense of national identity. Today, the date of the initial landings, 25 April, is a public holiday known as Anzac Day in Australia and New Zealand and every year thousands of people gather at memorials in both nations, and indeed in Turkey, to ...</i>	honour	0.995
<i>On the anniversary date of his death, every year since 1997, thousands of people gather at his home in Memphis to...</i>	celebrate	0.0086
<i>Twenty-five years after Marseille’s death, fighter pilot veterans of World War II gathered to...</i>	honour	0.0000041

Table 6: Another example where the  $k$ NN model places much higher probability mass on the correct target, compared to the LM. The nearest neighbors search has retrieved a training set context that is extremely similar to the test context, while very rare and in the long-tail of patterns.

Test Context ( $p_{kNN} = 0.959, p_{LM} = 0.503$ )	Test Target	
<i>U2 do what they’re best at, slipping into epic rock mode, playing music made for the arena”. In two other local newspaper reviews, critics praised the song’s inclusion in a sequence of greatest hits. For the PopMart Tour of 1997–...</i>	1998	
Training Set Context	Training Set Target	Context Probability
<i>Following their original intent, “Sunday Bloody Sunday” was not played during any of the forty-seven shows on the Lovetown Tour in 1989. The song reappeared for a brief period during the Zoo TV Tour, and late during the second half of PopMart Tour (1997–...</i>	1998	0.936
<i>They are 6 times Champions and they won the Challenge Cup in 1938, and have experienced two previous stretches in the Super League, 1997–...</i>	2002	0.0071
<i>About \$40 million (\$61.4 million in 2018 dollars) was spent on the property acquisition. After weather-related construction delays due to the El Nino season of the winter of 1997–...</i>	1998	0.0015
<i>This made it the highest-rated season of The X-Files to air as well as the highest rated Fox program for the 1997–...</i>	98	0.00000048

Table 7: In this example, the desired date pattern appears in many examples. Yet, the nearest neighbors search is able to identify the only training set context which is relevant to the test context and assigns it the highest probability mass.

<b>Test Context</b> ( $p_{\text{KNN}} = 0.624, p_{\text{LM}} = 0.167$ )	<b>Test Target</b>	
<i>Lord Strathcona awarded Gauthier a scholarship in 1906 that allowed her to return to Europe and continue her vocal studies. She returned there and continued both to study and give performances. Her first operatic performance came in 1909 in Pavia, Italy as Micaela in Bizet's...</i>		Carmen
<b>Training Set Context</b>	<b>Training Set Target</b>	<b>Context Probability</b>
<i>Despite poor relations with the orchestra, Mahler brought five new operas to the theatre, including Bizet's...</i>	Carmen	0.356
<i>The fourth movement of An die Jugend (1909), for instance, uses two of Niccolò Paganini's Caprices for solo violin (numbers 11 and 15), while the 1920 piece Piano Sonatina No. 6 (Fantasia da camera super Carmen) is based on themes from Georges Bizet's...</i>	opera	0.0937
<i>It also hosted the Ballet of her Majesty's Theatre in the mid-19th century, before returning to hosting the London premieres of such operas as Bizet's...</i>	Carmen	0.0686

Table 8: In this case, the model is able to memorize the fact that *Georges Bizet* wrote *Carmen*.

<b>Test Context</b> ( $p_{\text{KNN}} = 0.031, p_{\text{LM}} = 0.007$ )	<b>Test Target</b>	
<i>Mycena maculata bears some resemblance to <i>M. &lt;unk&gt;</i>, but is only associated with decaying hardwood logs and stumps, and is found in eastern North America, and sometimes on oak on the West Coast. In age, it...</i>		develops
<b>Training Set Context</b>	<b>Training Set Target</b>	<b>Context Probability</b>
<i>Morelula tridentina (=Morelula frustulata) is also rufescence and very similar to <i>M. rufobrunnea</i>. It is found in mountainous forests and maquis and forms a marked sinus at the attachment of the cap with the stem, which is pure white. At maturity, it...</i>	develops	0.031
<i>The winter bonnet (<i>M. tintinnabulum</i>) is a northern European species that is much smaller (cap diameter up to 2.6 cm (1.0 in) across) and has a brown cap, and has ragged hairs at the base. It...</i>	generally	0.029
<i>The "bleeding" will distinguish <i>Mycena atkinsoniana</i> from most other <i>Mycena</i> species commonly encountered. The common and widely distributed <i>M. sanguinolenta</i> is another "bleeder", but it is smaller than <i>M. atkinsoniana</i>, with a cap diameter ranging from 3 to 15 mm (0.1 to 0.6 in). Additionally, it...</i>	has	0.028
<i>Mycena flavoalba bears resemblance to some members of the genus <i>Hemimycena</i>, such as <i>H. lactea</i> and <i>H. &lt;unk&gt;</i>. It...</i>	can	0.018

Table 9: This is an example where the  $p_{\text{KNN}}$  distribution is relatively flat, as several words are plausible continuations. However, the nearest neighbors search assigns the highest probability to the correct target and a corresponding context that is particularly relevant. In contrast, the LM probability on the correct target is lower.

---

# Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

---

Patrick Lewis<sup>†‡</sup>, Ethan Perez<sup>\*</sup>,

Aleksandra Piktus<sup>†</sup>, Fabio Petroni<sup>†</sup>, Vladimir Karpukhin<sup>†</sup>, Naman Goyal<sup>†</sup>, Heinrich Küttler<sup>†</sup>,

Mike Lewis<sup>†</sup>, Wen-tau Yih<sup>†</sup>, Tim Rocktäschel<sup>†‡</sup>, Sebastian Riedel<sup>†‡</sup>, Douwe Kiela<sup>†</sup>

<sup>†</sup>Facebook AI Research; <sup>‡</sup>University College London; <sup>\*</sup>New York University;  
plewis@fb.com

## Abstract

Large pre-trained language models have been shown to store factual knowledge in their parameters, and achieve state-of-the-art results when fine-tuned on downstream NLP tasks. However, their ability to access and precisely manipulate knowledge is still limited, and hence on knowledge-intensive tasks, their performance lags behind task-specific architectures. Additionally, providing provenance for their decisions and updating their world knowledge remain open research problems. Pre-trained models with a differentiable access mechanism to explicit non-parametric memory have so far been only investigated for extractive downstream tasks. We explore a general-purpose fine-tuning recipe for retrieval-augmented generation (RAG) — models which combine pre-trained parametric and non-parametric memory for language generation. We introduce RAG models where the parametric memory is a pre-trained seq2seq model and the non-parametric memory is a dense vector index of Wikipedia, accessed with a pre-trained neural retriever. We compare two RAG formulations, one which conditions on the same retrieved passages across the whole generated sequence, and another which can use different passages per token. We fine-tune and evaluate our models on a wide range of knowledge-intensive NLP tasks and set the state of the art on three open domain QA tasks, outperforming parametric seq2seq models and task-specific retrieve-and-extract architectures. For language generation tasks, we find that RAG models generate more specific, diverse and factual language than a state-of-the-art parametric-only seq2seq baseline.

## 1 Introduction

Pre-trained neural language models have been shown to learn a substantial amount of in-depth knowledge from data [47]. They can do so without any access to an external memory, as a parameterized implicit knowledge base [51, 52]. While this development is exciting, such models do have downsides: They cannot easily expand or revise their memory, can't straightforwardly provide insight into their predictions, and may produce “hallucinations” [38]. Hybrid models that combine parametric memory with non-parametric (i.e., retrieval-based) memories [20, 26, 48] can address some of these issues because knowledge can be directly revised and expanded, and accessed knowledge can be inspected and interpreted. REALM [20] and ORQA [31], two recently introduced models that combine masked language models [8] with a differentiable retriever, have shown promising results,

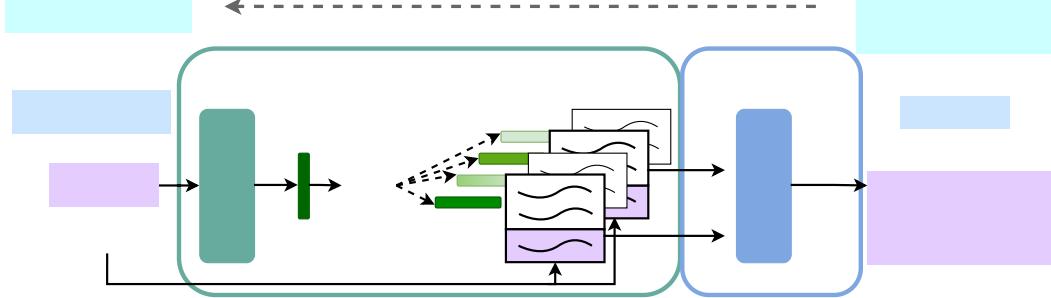


Figure 1: Overview of our approach. We combine a pre-trained retriever (*Query Encoder + Document Index*) with a pre-trained seq2seq model (*Generator*) and fine-tune end-to-end. For query  $x$ , we use Maximum Inner Product Search (MIPS) to find the top- $K$  documents  $z_i$ . For final prediction  $y$ , we treat  $z$  as a latent variable and marginalize over seq2seq predictions given different documents.

but have only explored open-domain extractive question answering. Here, we bring hybrid parametric and non-parametric memory to the “workhorse of NLP,” i.e. sequence-to-sequence (seq2seq) models.

We endow pre-trained, parametric-memory generation models with a non-parametric memory through a general-purpose fine-tuning approach which we refer to as retrieval-augmented generation (RAG). We build RAG models where the parametric memory is a pre-trained seq2seq transformer, and the non-parametric memory is a dense vector index of Wikipedia, accessed with a pre-trained neural retriever. We combine these components in a probabilistic model trained end-to-end (Fig. 1). The retriever (Dense Passage Retriever [26], henceforth DPR) provides latent documents conditioned on the input, and the seq2seq model (BART [32]) then conditions on these latent documents together with the input to generate the output. We marginalize the latent documents with a top- $K$  approximation, either on a per-output basis (assuming the same document is responsible for all tokens) or a per-token basis (where different documents are responsible for different tokens). Like T5 [51] or BART, RAG can be fine-tuned on any seq2seq task, whereby both the generator and retriever are jointly learned.

There has been extensive previous work proposing architectures to enrich systems with non-parametric memory which are trained from scratch for specific tasks, e.g. memory networks [64, 55], stack-augmented networks [25] and memory layers [30]. In contrast, we explore a setting where both parametric and non-parametric memory components are pre-trained and pre-loaded with extensive knowledge. Crucially, by using pre-trained access mechanisms, the ability to access knowledge is present without additional training.

Our results highlight the benefits of combining parametric and non-parametric memory with generation for *knowledge-intensive tasks*—tasks that humans could not reasonably be expected to perform without access to an external knowledge source. Our RAG models achieve state-of-the-art results on open Natural Questions [29], WebQuestions [3] and CuratedTrec [2] and strongly outperform recent approaches that use specialised pre-training objectives on TriviaQA [24]. Despite these being extractive tasks, we find that unconstrained generation outperforms previous extractive approaches. For knowledge-intensive generation, we experiment with MS-MARCO [1] and Jeopardy question generation, and we find that our models generate responses that are more factual, specific, and diverse than a BART baseline. For FEVER [56] fact verification, we achieve results within 4.3% of state-of-the-art pipeline models which use strong retrieval supervision. Finally, we demonstrate that the non-parametric memory can be replaced to update the models’ knowledge as the world changes<sup>1</sup>.

## 2 Methods

We explore RAG models, which use the input sequence  $x$  to retrieve text documents  $z$  and use them as additional context when generating the target sequence  $y$ . As shown in Figure 1, our models leverage two components: (i) a retriever  $p_\eta(z|x)$  with parameters  $\eta$  that returns (top- $K$  truncated) distributions over text passages given a query  $x$  and (ii) a generator  $p_\theta(y_i|x, z, y_{1:i-1})$  parametrized

<sup>1</sup>Code to run experiments with RAG has been open-sourced as part of the HuggingFace Transformers Library [66] and can be found at <https://github.com/huggingface/transformers/blob/master/examples/rag/>. An interactive demo of RAG models can be found at <https://huggingface.co/rag/>.

by  $\theta$  that generates a current token based on a context of the previous  $i - 1$  tokens  $y_{1:i-1}$ , the original input  $x$  and a retrieved passage  $z$ .

To train the retriever and generator end-to-end, we treat the retrieved document as a latent variable. We propose two models that marginalize over the latent documents in different ways to produce a distribution over generated text. In one approach, *RAG-Sequence*, the model uses the same document to predict each target token. The second approach, *RAG-Token*, can predict each target token based on a different document. In the following, we formally introduce both models and then describe the  $p_\eta$  and  $p_\theta$  components, as well as the training and decoding procedure.

## 2.1 Models

**RAG-Sequence Model** The RAG-Sequence model uses the same retrieved document to generate the complete *sequence*. Technically, it treats the retrieved document as a single latent variable that is marginalized to get the seq2seq probability  $p(y|x)$  via a top-K approximation. Concretely, the top K documents are retrieved using the retriever, and the generator produces the output sequence probability for each document, which are then marginalized,

$$p_{\text{RAG-Sequence}}(y|x) \approx \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y|x, z) = \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) \prod_i^N p_\theta(y_i|x, z, y_{1:i-1})$$

**RAG-Token Model** In the RAG-Token model we can draw a different latent document for each target *token* and marginalize accordingly. This allows the generator to choose content from several documents when producing an answer. Concretely, the top K documents are retrieved using the retriever, and then the generator produces a distribution for the next output token for each document, before marginalizing, and repeating the process with the following output token. Formally, we define:

$$p_{\text{RAG-Token}}(y|x) \approx \prod_i^N \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y_i|x, z, y_{1:i-1})$$

Finally, we note that RAG can be used for sequence classification tasks by considering the target class as a target sequence of length one, in which case RAG-Sequence and RAG-Token are equivalent.

## 2.2 Retriever: DPR

The retrieval component  $p_\eta(z|x)$  is based on DPR [26]. DPR follows a bi-encoder architecture:

$$p_\eta(z|x) \propto \exp(\mathbf{d}(z)^\top \mathbf{q}(x)) \quad \mathbf{d}(z) = \text{BERT}_d(z), \quad \mathbf{q}(x) = \text{BERT}_q(x)$$

where  $\mathbf{d}(z)$  is a dense representation of a document produced by a  $\text{BERT}_{\text{BASE}}$  *document encoder* [8], and  $\mathbf{q}(x)$  a query representation produced by a *query encoder*, also based on  $\text{BERT}_{\text{BASE}}$ . Calculating top-k( $p_\eta(\cdot|x)$ ), the list of  $k$  documents  $z$  with highest prior probability  $p_\eta(z|x)$ , is a Maximum Inner Product Search (MIPS) problem, which can be approximately solved in sub-linear time [23]. We use a pre-trained bi-encoder from DPR to initialize our retriever and to build the document index. This retriever was trained to retrieve documents which contain answers to TriviaQA [24] questions and Natural Questions [29]. We refer to the document index as the *non-parametric memory*.

## 2.3 Generator: BART

The generator component  $p_\theta(y_i|x, z, y_{1:i-1})$  could be modelled using any encoder-decoder. We use BART-large [32], a pre-trained seq2seq transformer [58] with 400M parameters. To combine the input  $x$  with the retrieved content  $z$  when generating from BART, we simply concatenate them. BART was pre-trained using a denoising objective and a variety of different noising functions. It has obtained state-of-the-art results on a diverse set of generation tasks and outperforms comparably-sized T5 models [32]. We refer to the BART generator parameters  $\theta$  as the *parametric memory* henceforth.

## 2.4 Training

We jointly train the retriever and generator components without any direct supervision on what document should be retrieved. Given a fine-tuning training corpus of input/output pairs  $(x_j, y_j)$ , we

minimize the negative marginal log-likelihood of each target,  $\sum_j -\log p(y_j|x_j)$  using stochastic gradient descent with Adam [28]. Updating the document encoder  $BERT_d$  during training is costly as it requires the document index to be periodically updated as REALM does during pre-training [20]. We do not find this step necessary for strong performance, and keep the document encoder (and index) fixed, only fine-tuning the query encoder  $BERT_q$  and the BART generator.

## 2.5 Decoding

At test time, RAG-Sequence and RAG-Token require different ways to approximate  $\arg \max_y p(y|x)$ .

**RAG-Token** The RAG-Token model can be seen as a standard, autoregressive seq2seq generator with transition probability:  $p'_\theta(y_i|x, y_{1:i-1}) = \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z_i|x)p_\theta(y_i|x, z_i, y_{1:i-1})$  To decode, we can plug  $p'_\theta(y_i|x, y_{1:i-1})$  into a standard beam decoder.

**RAG-Sequence** For RAG-Sequence, the likelihood  $p(y|x)$  does not break into a conventional per-token likelihood, hence we cannot solve it with a single beam search. Instead, we run beam search for each document  $z$ , scoring each hypothesis using  $p_\theta(y_i|x, z, y_{1:i-1})$ . This yields a set of hypotheses  $Y$ , some of which may not have appeared in the beams of all documents. To estimate the probability of an hypothesis  $y$  we run an additional forward pass for each document  $z$  for which  $y$  does not appear in the beam, multiply generator probability with  $p_\eta(z|x)$  and then sum the probabilities across beams for the marginals. We refer to this decoding procedure as “Thorough Decoding.” For longer output sequences,  $|Y|$  can become large, requiring many forward passes. For more efficient decoding, we can make a further approximation that  $p_\theta(y|x, z_i) \approx 0$  where  $y$  was not generated during beam search from  $x, z_i$ . This avoids the need to run additional forward passes once the candidate set  $Y$  has been generated. We refer to this decoding procedure as “Fast Decoding.”

## 3 Experiments

We experiment with RAG in a wide range of knowledge-intensive tasks. For all experiments, we use a single Wikipedia dump for our non-parametric knowledge source. Following Lee et al. [31] and Karpukhin et al. [26], we use the December 2018 dump. Each Wikipedia article is split into disjoint 100-word chunks, to make a total of 21M documents. We use the document encoder to compute an embedding for each document, and build a single MIPS index using FAISS [23] with a Hierarchical Navigable Small World approximation for fast retrieval [37]. During training, we retrieve the top  $k$  documents for each query. We consider  $k \in \{5, 10\}$  for training and set  $k$  for test time using dev data. We now discuss experimental details for each task.

### 3.1 Open-domain Question Answering

Open-domain question answering (QA) is an important real-world application and common testbed for knowledge-intensive tasks [20]. We treat questions and answers as input-output text pairs  $(x, y)$  and train RAG by directly minimizing the negative log-likelihood of answers. We compare RAG to the popular extractive QA paradigm [5, 7, 31, 26], where answers are extracted spans from retrieved documents, relying primarily on non-parametric knowledge. We also compare to “Closed-Book QA” approaches [52], which, like RAG, generate answers, but which do not exploit retrieval, instead relying purely on parametric knowledge. We consider four popular open-domain QA datasets: Natural Questions (NQ) [29], TriviaQA (TQA) [24], WebQuestions (WQ) [3] and CuratedTrec (CT) [2]. As CT and WQ are small, we follow DPR [26] by initializing CT and WQ models with our NQ RAG model. We use the same train/dev/test splits as prior work [31, 26] and report Exact Match (EM) scores. For TQA, to compare with T5 [52], we also evaluate on the TQA Wiki test set.

### 3.2 Abstractive Question Answering

RAG models can go beyond simple extractive QA and answer questions with free-form, abstractive text generation. To test RAG’s natural language generation (NLG) in a knowledge-intensive setting, we use the MSMARCO NLG task v2.1 [43]. The task consists of questions, ten gold passages retrieved from a search engine for each question, and a full sentence answer annotated from the retrieved passages. We do not use the supplied passages, only the questions and answers, to treat

MSMARCO as an open-domain abstractive QA task. MSMARCO has some questions that cannot be answered in a way that matches the reference answer without access to the gold passages, such as “What is the weather in Volcano, CA?” so performance will be lower without using gold passages. We also note that some MSMARCO questions cannot be answered using Wikipedia alone. Here, RAG can rely on parametric knowledge to generate reasonable responses.

### 3.3 Jeopardy Question Generation

To evaluate RAG’s generation abilities in a non-QA setting, we study open-domain question generation. Rather than use questions from standard open-domain QA tasks, which typically consist of short, simple questions, we propose the more demanding task of generating Jeopardy questions. Jeopardy is an unusual format that consists of trying to guess an entity from a fact about that entity. For example, “The World Cup” is the answer to the question “In 1986 Mexico scored as the first country to host this international sports competition twice.” As Jeopardy questions are precise, factual statements, generating Jeopardy questions conditioned on their answer entities constitutes a challenging knowledge-intensive generation task.

We use the splits from SearchQA [10], with 100K train, 14K dev, and 27K test examples. As this is a new task, we train a BART model for comparison. Following [67], we evaluate using the SQuAD-tuned Q-BLEU-1 metric [42]. Q-BLEU is a variant of BLEU with a higher weight for matching entities and has higher correlation with human judgment for question generation than standard metrics. We also perform two human evaluations, one to assess generation factuality, and one for specificity. We define factuality as whether a statement can be corroborated by trusted external sources, and specificity as high mutual dependence between the input and output [33]. We follow best practice and use pairwise comparative evaluation [34]. Evaluators are shown an answer and two generated questions, one from BART and one from RAG. They are then asked to pick one of four options—question A is better, question B is better, both are good, or neither is good.

### 3.4 Fact Verification

FEVER [56] requires classifying whether a natural language claim is supported or refuted by Wikipedia, or whether there is not enough information to decide. The task requires retrieving evidence from Wikipedia relating to the claim and then reasoning over this evidence to classify whether the claim is true, false, or unverifiable from Wikipedia alone. FEVER is a retrieval problem coupled with an challenging entailment reasoning task. It also provides an appropriate testbed for exploring the RAG models’ ability to handle classification rather than generation. We map FEVER class labels (supports, refutes, or not enough info) to single output tokens and directly train with claim-class pairs. Crucially, unlike most other approaches to FEVER, we do not use supervision on retrieved evidence. In many real-world applications, retrieval supervision signals aren’t available, and models that do not require such supervision will be applicable to a wider range of tasks. We explore two variants: the standard 3-way classification task (supports/refutes/not enough info) and the 2-way (supports/refutes) task studied in Thorne and Vlachos [57]. In both cases we report label accuracy.

## 4 Results

### 4.1 Open-domain Question Answering

Table I shows results for RAG along with state-of-the-art models. On all four open-domain QA tasks, RAG sets a new state of the art (only on the T5-comparable split for TQA). RAG combines the generation flexibility of the “closed-book” (parametric only) approaches and the performance of “open-book” retrieval-based approaches. Unlike REALM and T5+SSM, RAG enjoys strong results without expensive, specialized “salient span masking” pre-training [20]. It is worth noting that RAG’s retriever is initialized using DPR’s retriever, which uses retrieval supervision on Natural Questions and TriviaQA. RAG compares favourably to the DPR QA system, which uses a BERT-based “cross-encoder” to re-rank documents, along with an extractive reader. RAG demonstrates that neither a re-ranker nor extractive reader is necessary for state-of-the-art performance.

There are several advantages to generating answers even when it is possible to extract them. Documents with clues about the answer but do not contain the answer verbatim can still contribute towards a correct answer being generated, which is not possible with standard extractive approaches, leading

Table 1: Open-Domain QA Test Scores. For TQA, left column uses the standard test set for Open-Domain QA, right column uses the TQA-Wiki test set. See Appendix D for further details.

	Model	NQ	TQA	WQ	CT
Closed Book	T5-11B [52]	34.5	- /50.1	37.4	-
	T5-11B+SSM [52]	36.6	- /60.5	44.7	-
Open Book	REALM [20]	40.4	- / -	40.7	46.8
	DPR [26]	41.5	<b>57.9/</b> -	41.1	50.6
	RAG-Token	44.1	55.2/66.1	<b>45.5</b>	50.0
	RAG-Seq.	<b>44.5</b>	56.8/ <b>68.0</b>	45.2	<b>52.2</b>

Table 2: Generation and classification Test Scores. MS-MARCO SotA is [4], FEVER-3 is [68] and FEVER-2 is [57]. \*Uses gold context/evidence. Best model without gold access underlined.

Model	Jeopardy		MSMARCO		FVR3	FVR2
	B-1	QB-1	R-L	B-1	Label Acc.	
SotA	-	-	<b>49.8*</b>	<b>49.9*</b>	<b>76.8</b>	<b>92.2*</b>
BART	15.1	19.7	38.2	41.6	64.0	81.1
RAG-Tok.	<b>17.3</b>	<b>22.2</b>	40.1	41.5	72.5	<u>89.5</u>
RAG-Seq.	14.7	21.4	<u>40.8</u>	<u>44.2</u>		

to more effective marginalization over documents. Furthermore, RAG can generate correct answers even when the correct answer is not in any retrieved document, achieving 11.8% accuracy in such cases for NQ, where an extractive model would score 0%.

## 4.2 Abstractive Question Answering

As shown in Table 2, RAG-Sequence outperforms BART on Open MS-MARCO NLG by 2.6 Bleu points and 2.6 Rouge-L points. RAG approaches state-of-the-art model performance, which is impressive given that (i) those models access gold passages with specific information required to generate the reference answer, (ii) many questions are unanswerable without the gold passages, and (iii) not all questions are answerable from Wikipedia alone. Table 3 shows some generated answers from our models. Qualitatively, we find that RAG models hallucinate less and generate factually correct text more often than BART. Later, we also show that RAG generations are more diverse than BART generations (see §4.5).

## 4.3 Jeopardy Question Generation

Table 2 shows that RAG-Token performs better than RAG-Sequence on Jeopardy question generation, with both models outperforming BART on Q-BLEU-1. 4 shows human evaluation results, over 452 pairs of generations from BART and RAG-Token. Evaluators indicated that BART was more factual than RAG in only 7.1% of cases, while RAG was more factual in 42.7% of cases, and both RAG and BART were factual in a further 17% of cases, clearly demonstrating the effectiveness of RAG on the task over a state-of-the-art generation model. Evaluators also find RAG generations to be more specific by a large margin. Table 3 shows typical generations from each model.

Jeopardy questions often contain two separate pieces of information, and RAG-Token may perform best because it can generate responses that combine content from several documents. Figure 2 shows an example. When generating “Sun”, the posterior is high for document 2 which mentions “The Sun Also Rises”. Similarly, document 1 dominates the posterior when “A Farewell to Arms” is generated. Intriguingly, after the first token of each book is generated, the document posterior flattens. This observation suggests that the generator can complete the titles without depending on specific documents. In other words, the model’s parametric knowledge is sufficient to complete the titles. We find evidence for this hypothesis by feeding the BART-only baseline with the partial decoding "The Sun. BART completes the generation "The Sun Also Rises" is a novel by this author of "The Sun Also Rises" indicating the title "The Sun Also Rises" is stored in BART’s parameters. Similarly, BART will complete the partial decoding "The Sun Also Rises" is a novel by this author of "A with "The Sun Also Rises" is a novel by this author of "A Farewell to Arms". This example shows how parametric and non-parametric memories work together—the non-parametric component helps to guide the generation, drawing out specific knowledge stored in the parametric memory.

## 4.4 Fact Verification

Table 2 shows our results on FEVER. For 3-way classification, RAG scores are within 4.3% of state-of-the-art models, which are complex pipeline systems with domain-specific architectures and substantial engineering, trained using intermediate retrieval supervision, which RAG does not require.

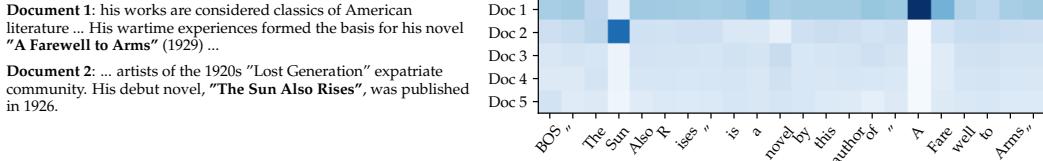


Figure 2: RAG-Token document posterior  $p(z_i|x, y_i, y_{-i})$  for each generated token for input "Hemingway" for Jeopardy generation with 5 retrieved documents. The posterior for document 1 is high when generating "A Farewell to Arms" and for document 2 when generating "The Sun Also Rises".

Table 3: Examples from generation tasks. RAG models generate more specific and factually accurate responses. '?' indicates factually incorrect responses, '\*' indicates partially correct responses.

Task	Input	Model	Generation
MS-MARCO	define middle ear	BART	?The middle ear is the part of the ear between the middle ear and the nose.
		RAG-T	The middle ear is the portion of the ear internal to the eardrum.
		RAG-S	The middle ear includes the tympanic cavity and the three ossicles.
Jeopardy Question Generation	what currency needed in scotland	BART	The currency needed in Scotland is Pound sterling.
		RAG-T	Pound is the currency needed in Scotland.
		RAG-S	The currency needed in Scotland is the pound sterling.
Jeopardy Question Generation	Washington	BART	?This state has the largest number of counties in the U.S.
		RAG-T	It's the only U.S. state named for a U.S. president
		RAG-S	It's the state where you'll find Mount Rainier National Park
RAG-S	The Divine Comedy	BART	*This epic poem by Dante is divided into 3 parts: the Inferno, the Purgatorio & the Purgatorio
		RAG-T	Dante's "Inferno" is the first part of this epic poem
		RAG-S	This 14th century work is divided into 3 sections: "Inferno", "Purgatorio" & "Paradiso"

For 2-way classification, we compare against Thorne and Vlachos [57], who train RoBERTa [35] to classify the claim as true or false given the gold evidence sentence. RAG achieves an accuracy within 2.7% of this model, despite being supplied with only the claim and retrieving its own evidence. We also analyze whether documents retrieved by RAG correspond to documents annotated as gold evidence in FEVER. We calculate the overlap in article titles between the top  $k$  documents retrieved by RAG and gold evidence annotations. We find that the top retrieved document is from a gold article in 71% of cases, and a gold article is present in the top 10 retrieved articles in 90% of cases.

#### 4.5 Additional Results

**Generation Diversity** Section 4.3 shows that RAG models are more factual and specific than BART for Jeopardy question generation. Following recent work on diversity-promoting decoding [33, 59, 39], we also investigate generation diversity by calculating the ratio of distinct ngrams to total ngrams generated by different models. Table 5 shows that RAG-Sequence's generations are more diverse than RAG-Token's, and both are significantly more diverse than BART without needing any diversity-promoting decoding.

**Retrieval Ablations** A key feature of RAG is learning to retrieve relevant information for the task. To assess the effectiveness of the retrieval mechanism, we run ablations where we freeze the retriever during training. As shown in Table 6 learned retrieval improves results for all tasks.

We compare RAG's dense retriever to a word overlap-based BM25 retriever [53]. Here, we replace RAG's retriever with a fixed BM25 system, and use BM25 retrieval scores as logits when calculating  $p(z|x)$ . Table 6 shows the results. For FEVER, BM25 performs best, perhaps since FEVER claims are heavily entity-centric and thus well-suited for word overlap-based retrieval. Differentiable retrieval improves results on all other tasks, especially for Open-Domain QA, where it is crucial.

**Index hot-swapping** An advantage of non-parametric memory models like RAG is that knowledge can be easily updated at test time. Parametric-only models like T5 or BART need further training to update their behavior as the world changes. To demonstrate, we build an index using the DrQA [5] Wikipedia dump from December 2016 and compare outputs from RAG using this index to the newer index from our main results (December 2018). We prepare a list of 82 world leaders who had changed

Table 4: Human assessments for the Jeopardy Question Generation Task.

	Factuality	Specificity
BART better	7.1%	16.8%
RAG better	<b>42.7%</b>	<b>37.4%</b>
Both good	11.7%	11.8%
Both poor	17.7%	6.9%
No majority	20.8%	20.1%

Table 5: Ratio of distinct to total tri-grams for generation tasks.

	MSMARCO	Jeopardy QGen
Gold	89.6%	90.0%
BART	70.7%	32.4%
RAG-Token	77.8%	46.8%
RAG-Seq.	83.5%	53.8%

Table 6: Ablations on the dev set. As FEVER is a classification task, both RAG models are equivalent.

Model	NQ	TQA	WQ	CT	Jeopardy-QGen	MSMarco	FVR-3	FVR-2
	Exact	Match			B-1	R-L	B-1	Label Accuracy
RAG-Token-BM25	29.7	41.5	32.1	33.1	17.5	22.3	55.5	48.4
RAG-Sequence-BM25	31.8	44.1	36.6	33.8	11.1	19.5	56.5	46.9
RAG-Token-Frozen	37.8	50.1	37.1	51.1	16.7	21.7	55.9	49.4
RAG-Sequence-Frozen	41.2	52.1	41.8	52.6	11.8	19.6	56.7	47.3
RAG-Token	43.5	54.8	<b>46.5</b>	51.9	<b>17.9</b>	<b>22.6</b>	56.2	<b>49.4</b>
RAG-Sequence	<b>44.0</b>	<b>55.8</b>	44.9	<b>53.4</b>	15.3	21.5	<b>57.2</b>	47.5

between these dates and use a template “Who is {position}?” (e.g. “Who is the President of Peru?”) to query our NQ RAG model with each index. RAG answers 70% correctly using the 2016 index for 2016 world leaders and 68% using the 2018 index for 2018 world leaders. Accuracy with mismatched indices is low (12% with the 2018 index and 2016 leaders, 4% with the 2016 index and 2018 leaders). This shows we can update RAG’s world knowledge by simply replacing its non-parametric memory.

**Effect of Retrieving more documents** Models are trained with either 5 or 10 retrieved latent documents, and we do not observe significant differences in performance between them. We have the flexibility to adjust the number of retrieved documents at test time, which can affect performance and runtime. Figure 3(left) shows that retrieving more documents at test time monotonically improves Open-domain QA results for RAG-Sequence, but performance peaks for RAG-Token at 10 retrieved documents. Figure 3(right) shows that retrieving more documents leads to higher Rouge-L for RAG-Token at the expense of Bleu-1, but the effect is less pronounced for RAG-Sequence.

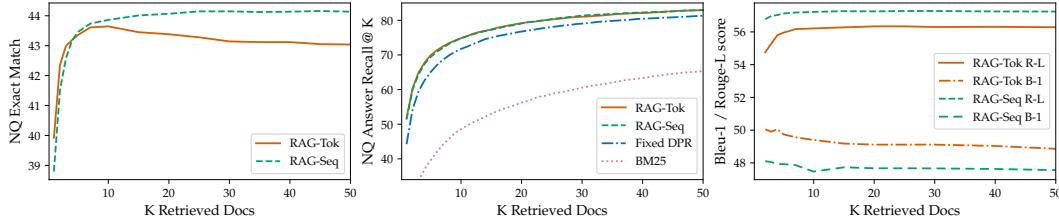


Figure 3: Left: NQ performance as more documents are retrieved. Center: Retrieval recall performance in NQ. Right: MS-MARCO Bleu-1 and Rouge-L as more documents are retrieved.

## 5 Related Work

**Single-Task Retrieval** Prior work has shown that retrieval improves performance across a variety of NLP tasks when considered in isolation. Such tasks include open-domain question answering [5, 29], fact checking [56], fact completion [48], long-form question answering [12], Wikipedia article generation [36], dialogue [41, 65, 9, 13], translation [17], and language modeling [19, 27]. Our work unifies previous successes in incorporating retrieval into individual tasks, showing that a single retrieval-based architecture is capable of achieving strong performance across several tasks.

**General-Purpose Architectures for NLP** Prior work on general-purpose architectures for NLP tasks has shown great success without the use of retrieval. A single, pre-trained language model has been shown to achieve strong performance on various classification tasks in the GLUE benchmarks [60, 61] after fine-tuning [49, 8]. GPT-2 [50] later showed that a single, left-to-right, pre-trained language model could achieve strong performance across both discriminative and generative tasks. For further improvement, BART [32] and T5 [51, 52] propose a single, pre-trained encoder-decoder model that leverages bi-directional attention to achieve stronger performance on discriminative and generative tasks. Our work aims to expand the space of possible tasks with a single, unified architecture, by learning a retrieval module to augment pre-trained, generative language models.

**Learned Retrieval** There is significant work on learning to retrieve documents in information retrieval, more recently with pre-trained, neural language models [44, 26] similar to ours. Some work optimizes the retrieval module to aid in a specific, downstream task such as question answering, using search [46], reinforcement learning [6, 63, 62], or a latent variable approach [31, 20] as in our work. These successes leverage different retrieval-based architectures and optimization techniques to achieve strong performance on a single task, while we show that a single retrieval-based architecture can be fine-tuned for strong performance on a variety of tasks.

**Memory-based Architectures** Our document index can be seen as a large external memory for neural networks to attend to, analogous to memory networks [64, 55]. Concurrent work [14] learns to retrieve a trained embedding for each entity in the input, rather than to retrieve raw text as in our work. Other work improves the ability of dialog models to generate factual text by attending over fact embeddings [15, 13]. A key feature of our memory is that it is comprised of raw text rather than distributed representations, which makes the memory both (i) human-readable, lending a form of interpretability to our model, and (ii) human-writable, enabling us to dynamically update the model’s memory by editing the document index. This approach has also been used in knowledge-intensive dialog, where generators have been conditioned on retrieved text directly, albeit obtained via TF-IDF rather than end-to-end learnt retrieval [9].

**Retrieve-and-Edit approaches** Our method shares some similarities with retrieve-and-edit style approaches, where a similar training input-output pair is retrieved for a given input, and then edited to provide a final output. These approaches have proved successful in a number of domains including Machine Translation [18, 22] and Semantic Parsing [21]. Our approach does have several differences, including less of emphasis on lightly editing a retrieved item, but on aggregating content from several pieces of retrieved content, as well as learning latent retrieval, and retrieving evidence documents rather than related training pairs. This said, RAG techniques may work well in these settings, and could represent promising future work.

## 6 Discussion

In this work, we presented hybrid generation models with access to parametric and non-parametric memory. We showed that our RAG models obtain state of the art results on open-domain QA. We found that people prefer RAG’s generation over purely parametric BART, finding RAG more factual and specific. We conducted an thorough investigation of the learned retrieval component, validating its effectiveness, and we illustrated how the retrieval index can be hot-swapped to update the model without requiring any retraining. In future work, it may be fruitful to investigate if the two components can be jointly pre-trained from scratch, either with a denoising objective similar to BART or some another objective. Our work opens up new research directions on how parametric and non-parametric memories interact and how to most effectively combine them, showing promise in being applied to a wide variety of NLP tasks.

## Broader Impact

This work offers several positive societal benefits over previous work: the fact that it is more strongly grounded in real factual knowledge (in this case Wikipedia) makes it “hallucinate” less with generations that are more factual, and offers more control and interpretability. RAG could be employed in a wide variety of scenarios with direct benefit to society, for example by endowing it with a medical index and asking it open-domain questions on that topic, or by helping people be more effective at their jobs.

With these advantages also come potential downsides: Wikipedia, or any potential external knowledge source, will probably never be entirely factual and completely devoid of bias. Since RAG can be employed as a language model, similar concerns as for GPT-2 [50] are valid here, although arguably to a lesser extent, including that it might be used to generate abuse, faked or misleading content in the news or on social media; to impersonate others; or to automate the production of spam/phishing content [54]. Advanced language models may also lead to the automation of various jobs in the coming decades [16]. In order to mitigate these risks, AI systems could be employed to fight against misleading content and automated spam/phishing.

## Acknowledgments

The authors would like to thank the reviewers for their thoughtful and constructive feedback on this paper, as well as HuggingFace for their help in open-sourcing code to run RAG models. The authors would also like to thank Kyunghyun Cho and Sewon Min for productive discussions and advice. EP thanks supports from the NSF Graduate Research Fellowship. PL is supported by the FAIR PhD program.

## References

- [1] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. MS MARCO: A Human Generated MAchine Reading COnprehension Dataset. *arXiv:1611.09268 [cs]*, November 2016. URL <http://arxiv.org/abs/1611.09268>. arXiv: 1611.09268.
- [2] Petr Baudiš and Jan Šedivý. Modeling of the question answering task in the yodaqa system. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 222–228. Springer, 2015. URL [https://link.springer.com/chapter/10.1007/978-3-319-24027-5\\_20](https://link.springer.com/chapter/10.1007/978-3-319-24027-5_20).
- [3] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/D13-1160>
- [4] Bin Bi, Chenliang Li, Chen Wu, Ming Yan, and Wei Wang. Palm: Pre-training an autoencoding&autoregressive language model for context-conditioned generation. *ArXiv*, abs/2004.07159, 2020. URL <https://arxiv.org/abs/2004.07159>.
- [5] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to Answer Open-Domain Questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1171. URL <https://www.aclweb.org/anthology/P17-1171>.
- [6] Eunsol Choi, Daniel Hewlett, Jakob Uszkoreit, Illia Polosukhin, Alexandre Lacoste, and Jonathan Berant. Coarse-to-fine question answering for long documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 209–220, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1020. URL <https://www.aclweb.org/anthology/P17-1020>.

# Appendices for Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

## A Implementation Details

For Open-domain QA we report test numbers using 15 retrieved documents for RAG-Token models. For RAG-Sequence models, we report test results using 50 retrieved documents, and we use the Thorough Decoding approach since answers are generally short. We use greedy decoding for QA as we did not find beam search improved results. For Open-MSMarco and Jeopardy question generation, we report test numbers using ten retrieved documents for both RAG-Token and RAG-Sequence, and we also train a BART-large model as a baseline. We use a beam size of four, and use the Fast Decoding approach for RAG-Sequence models, as Thorough Decoding did not improve performance.

## B Human Evaluation

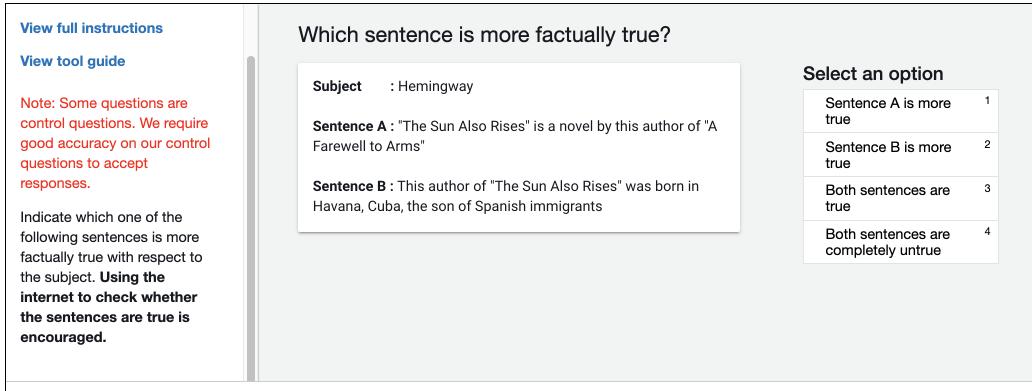


Figure 4: Annotation interface for human evaluation of factuality. A pop-out for detailed instructions and a worked example appear when clicking "view tool guide".

Figure 4 shows the user interface for human evaluation. To avoid any biases for screen position, which model corresponded to sentence A and sentence B was randomly selected for each example. Annotators were encouraged to research the topic using the internet, and were given detailed instructions and worked examples in a full instructions tab. We included some gold sentences in order to assess the accuracy of the annotators. Two annotators did not perform well on these examples and their annotations were removed from the results.

## C Training setup Details

We train all RAG models and BART baselines using Fairseq [45].<sup>2</sup> We train with mixed precision floating point arithmetic [40], distributing training across 8, 32GB NVIDIA V100 GPUs, though training and inference can be run on one GPU. We find that doing Maximum Inner Product Search with FAISS is sufficiently fast on CPU, so we store document index vectors on CPU, requiring  $\sim 100$  GB of CPU memory for all of Wikipedia. After submission, We have ported our code to HuggingFace Transformers [66]<sup>3</sup>, which achieves equivalent performance to the previous version but is a cleaner and easier to use implementation. This version is also open-sourced. We also compress the document index using FAISS's compression tools, reducing the CPU memory requirement to 36GB. Scripts to run experiments with RAG can be found at <https://github.com/huggingface/transformers/blob/master/examples/rag/README.md> and an interactive demo of a RAG model can be found at <https://huggingface.co/rag/>

<sup>2</sup><https://github.com/pytorch/fairseq>

<sup>3</sup><https://github.com/huggingface/transformers>

## D Further Details on Open-Domain QA

For open-domain QA, multiple answer annotations are often available for a given question. These answer annotations are exploited by extractive models during training as typically all the answer annotations are used to find matches within documents when preparing training data. For RAG, we also make use of multiple annotation examples for Natural Questions and WebQuestions by training the model with each  $(q, a)$  pair separately, leading to a small increase in accuracy. For TriviaQA, there are often many valid answers to a given question, some of which are not suitable training targets, such as emoji or spelling variants. For TriviaQA, we filter out answer candidates if they do not occur in top 1000 documents for the query.

**CuratedTrec preprocessing** The answers for CuratedTrec are given in the form of regular expressions, which has been suggested as a reason why it is unsuitable for answer-generation models [20]. To overcome this, we use a pre-processing step where we first retrieve the top 1000 documents for each query, and use the answer that most frequently matches the regex pattern as the supervision target. If no matches are found, we resort to a simple heuristic: generate all possible permutations for each regex, replacing non-deterministic symbols in the regex nested tree structure with a whitespace.

**TriviaQA Evaluation setups** The open-domain QA community customarily uses public development datasets as test datasets, as test data for QA datasets is often restricted and dedicated to reading comprehension purposes. We report our results using the datasets splits used in DPR [26], which are consistent with common practice in Open-domain QA. For TriviaQA, this test dataset is the public TriviaQA Web Development split. Roberts et al. [52] used the TriviaQA official Wikipedia test set instead. Févry et al. [14] follow this convention in order to compare with Roberts et al. [52] (See appendix of [14]). We report results on both test sets to enable fair comparison to both approaches. We find that our performance is much higher using the official Wiki test set, rather than the more conventional open-domain test set, which we attribute to the official Wiki test set questions being simpler to answer from Wikipedia.

## E Further Details on FEVER

For FEVER classification, we follow the practice from [32], and first re-generate the claim, and then classify using the representation of the final hidden state, before finally marginalizing across documents to obtain the class probabilities. The FEVER task traditionally has two sub-tasks. The first is to classify the claim as either "Supported", "Refuted" or "Not Enough Info", which is the task we explore in the main paper. FEVER's other sub-task involves extracting sentences from Wikipedia as evidence supporting the classification prediction. As FEVER uses a different Wikipedia dump to us, directly tackling this task is not straightforward. We hope to address this in future work.

## F Null Document Probabilities

We experimented with adding "Null document" mechanism to RAG, similar to REALM [20] in order to model cases where no useful information could be retrieved for a given input. Here, if  $k$  documents were retrieved, we would additionally "retrieve" an empty document and predict a logit for the null document, before marginalizing over  $k + 1$  predictions. We explored modelling this null document logit by learning (i) a document embedding for the null document, (ii) a static learnt bias term, or (iii) a neural network to predict the logit. We did not find that these improved performance, so in the interests of simplicity, we omit them. For Open MS-MARCO, where useful retrieved documents cannot always be retrieved, we observe that the model learns to always retrieve a particular set of documents for questions that are less likely to benefit from retrieval, suggesting that null document mechanisms may not be necessary for RAG.

## G Parameters

Our RAG models contain the trainable parameters for the BERT-base query and document encoder of DPR, with 110M parameters each (although we do not train the document encoder ourselves) and 406M trainable parameters from BART-large, 406M parameters, making a total of 626M trainable

Table 7: Number of instances in the datasets used. \*A hidden subset of this data is used for evaluation

Task	Train	Development	Test
Natural Questions	79169	8758	3611
TriviaQA	78786	8838	11314
WebQuestions	3418	362	2033
CuratedTrec	635	134	635
Jeopardy Question Generation	97392	13714	26849
MS-MARCO	153726	12468	101093*
FEVER-3-way	145450	10000	10000
FEVER-2-way	96966	6666	6666

parameters. The best performing "closed-book" (parametric only) open-domain QA model is T5-11B with 11 Billion trainable parameters. The T5 model with the closest number of parameters to our models is T5-large (770M parameters), which achieves a score of 28.9 EM on Natural Questions [52], substantially below the 44.5 that RAG-Sequence achieves, indicating that hybrid parametric/non-parametric models require far fewer trainable parameters for strong open-domain QA performance. The non-parametric memory index does not consist of trainable parameters, but does consists of 21M 728 dimensional vectors, consisting of 15.3B values. These can be easily be stored at 8-bit floating point precision to manage memory and disk footprints.

## H Retrieval Collapse

In preliminary experiments, we observed that for some tasks such as story generation [11], the retrieval component would "collapse" and learn to retrieve the same documents regardless of the input. In these cases, once retrieval had collapsed, the generator would learn to ignore the documents, and the RAG model would perform equivalently to BART. The collapse could be due to a less-explicit requirement for factual knowledge in some tasks, or the longer target sequences, which could result in less informative gradients for the retriever. Perez et al. [46] also found spurious retrieval results when optimizing a retrieval component in order to improve performance on downstream tasks.

## I Number of instances per dataset

The number of training, development and test datapoints in each of our datasets is shown in Table 7

# Improving language models by retrieving from trillions of tokens

Sebastian Borgeaud<sup>†</sup>, Arthur Mensch<sup>†</sup>, Jordan Hoffmann<sup>†</sup>, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae<sup>‡</sup>, Erich Elsen<sup>‡</sup> and Laurent Sifre<sup>†,‡</sup>

All authors from DeepMind, <sup>†</sup>Equal contributions, <sup>‡</sup>Equal senior authorship

We enhance auto-regressive language models by conditioning on document chunks retrieved from a large corpus, based on local similarity with preceding tokens. With a 2 trillion token database, our Retrieval-Enhanced Transformer (**RETRO**) obtains comparable performance to GPT-3 and Jurassic-1 on the Pile, despite using  $25\times$  fewer parameters. After fine-tuning, **RETRO** performance translates to downstream knowledge-intensive tasks such as question answering. **RETRO** combines a frozen **BERT** retriever, a differentiable encoder and a chunked cross-attention mechanism to predict tokens based on an order of magnitude more data than what is typically consumed during training. We typically train **RETRO** from scratch, yet can also rapidly **RETROfit** pre-trained transformers with retrieval and still achieve good performance. Our work opens up new avenues for improving language models through explicit memory at unprecedented scale.

## 1. Introduction

Language modelling (LM) is an unsupervised task that consists of modelling the probability of text, usually by factorising it into conditional next-token predictions  $p(x_1, \dots, x_n) = \prod_i p(x_i|x_{<i})$ . Neural networks have proven to be powerful language models, first in the form of recurrent architectures (Graves, 2013; Jozefowicz et al., 2016; Mikolov et al., 2010) and more recently in the form of Transformers (Vaswani et al., 2017), that use attention to contextualise the past. Large performance improvements have come from increasing the amount of data, training compute, or model parameters. Transformers have been scaled from 100 million parameter models in seminal work to over hundred billion parameters (Brown et al., 2020; Radford et al., 2019) in the last two years which has led to models that do very well on a wide array of tasks in a zero or few-shot formulation. Increasing model size predictably improves performance on a wide range of downstream tasks (Kaplan et al., 2020). The benefits of increasing the number of parameters come from two factors: additional computations at training and inference time, and increased memorization of the training data.

In this work, we endeavor to decouple these, by exploring efficient means of augmenting language models with a massive-scale memory without significantly increasing computations. Specifically, we suggest retrieval from a large text database as a complementary path to scaling language models. Instead of increasing the size of the model and training on more data, we equip models with the ability to directly access a large database to perform predictions—a semi-parametric approach. At a high level, our Retrieval Transformer (**RETRO**) model splits the input sequence into chunks and retrieves text similar to the previous chunk to improve the predictions in the current chunk. Existing retrieval for language modelling work only considers small transformers (100 millions parameters) and databases of limited size (up to billions of tokens) (Guu et al., 2020; Khandelwal et al., 2020; Lewis et al., 2020; Yogatama et al., 2021). To our knowledge, our work is the first to show the benefits of scaling the retrieval database to trillions of tokens for large parametric language models. Our main

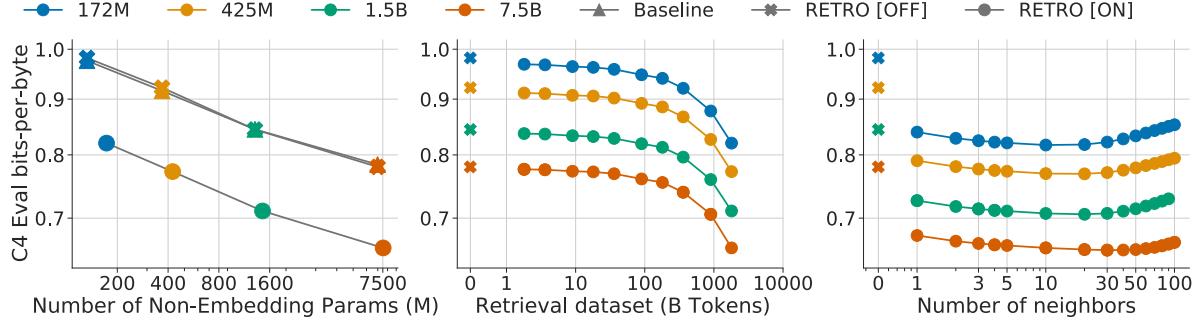


Figure 1 | **Scaling of RETRO.** The performance gain of our retrieval models remains constant with model scale (left), and is comparable to multiplying the parametric model size by  $\sim 10\times$ . The gain increases with the size of the retrieval database (middle) and the number of retrieved neighbours (right) on the C4 validation set, when using up to 40 neighbours. Past this, performance begins to degrade, perhaps due to the reduced quality. At evaluation RETRO can be used without retrieval data (RETRO [OFF]), bringing limited performance degradation compared to baseline transformers.

contributions are the following.

- We introduce RETRO, a retrieval-enhanced autoregressive language model (§2.2). We use a chunked cross-attention module to incorporate the retrieved text (§2.4), with time complexity linear in the amount of retrieved data. We show that retrieving based on a pre-trained frozen BERT model (§2.3) works at scale, removing the need for training and updating a retriever network.
- We show that our method scales well with model size and database size (Fig. 1): RETRO provides a constant gain for models ranging from 150M to 7B parameters, and RETRO can be improved at evaluation time by increasing the database size and the number of retrieved neighbours. Our largest model obtains state-of-the-art results on a range of downstream evaluation datasets including WikiText103 (Merity et al., 2017) and the Pile (Gao et al., 2020) (§4). We show that RETRO can be fine-tuned to achieve competitive performance on downstream tasks such as question answering (§4.3).
- We propose an evaluation aware of proximity of test documents with the training set (§2.6), addressing the problem of test set leakage (Lee et al., 2021). This is relevant for all language models, and especially for retrieval-enhanced models since they have direct access to the training dataset during evaluation. Using this methodology, we show that the performance of RETRO comes from both explicit neighbour copying and general knowledge extraction (§4.4).

## 2. Method

We design our retrieval-enhanced architecture to be capable of retrieving from a database with trillions of tokens. For this purpose, we retrieve at the level of contiguous token *chunks* instead of individual tokens which reduces storage and computation requirements by a large linear factor. Our method first constructs a key-value database, where values store raw chunks of text tokens and keys are frozen BERT embeddings (Devlin et al., 2019). We use a frozen model to avoid having to periodically re-compute embeddings over the entire database during training. Each training sequence is then split into chunks, which are augmented with their  $k$ -nearest neighbour retrieved from the database. An encoder-decoder architecture integrates retrieval chunks into the model’s predictions. We summarize the RETRO architecture in Fig. 2, and detail it in this section. We end the section by introducing

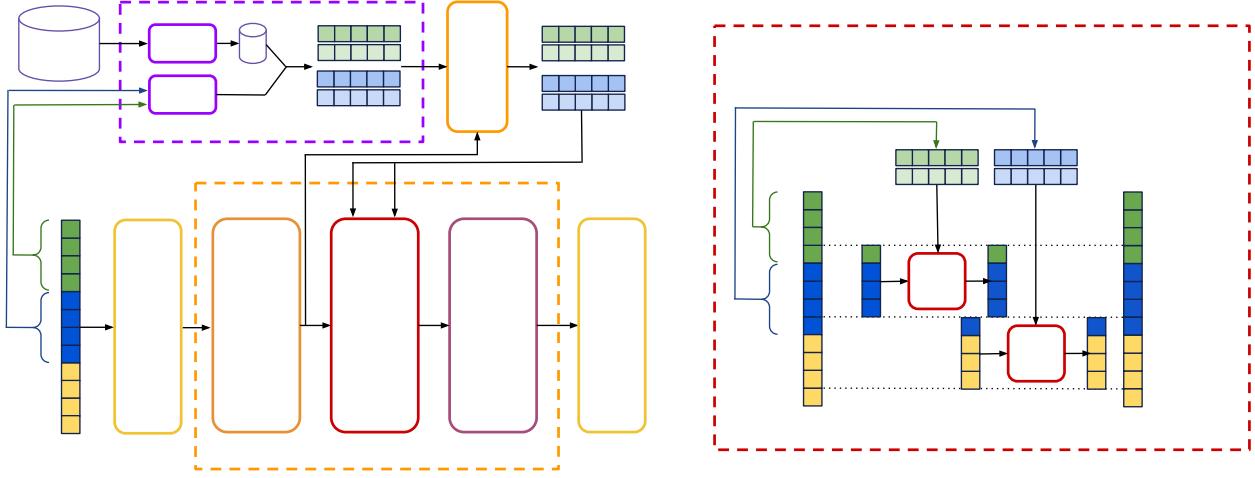


Figure 2 | **RETRO** architecture. *Left:* simplified version where a sequence of length  $n = 12$  is split into  $l = 3$  chunks of size  $m = 4$ . For each chunk, we retrieve  $k = 2$  neighbours of  $r = 5$  tokens each. The retrieval pathway is shown on top. *Right:* Details of the interactions in the CCA operator. Causality is maintained as neighbours of the first chunk only affect the last token of the first chunk and tokens from the second chunk.

a new methodology to evaluate language models when an evaluation set is partially present in the training set.

## 2.1. Training dataset

We use a multi-lingual version of *MassiveText* (Rae et al., 2021) for both training and retrieval data. The dataset consists of text documents from multiple sources and multiple languages totalling over 5 trillion tokens (detailed in Table 1). Sequences are sampled from subsets of the training data, with sampling weights given in the right-most column of Table 1. We tokenize the dataset using SentencePiece (Kudo and Richardson, 2018) with a vocabulary of 128,000 tokens. During training (unless otherwise specified), we retrieve from 600B tokens from the training data. The training retrieval database is made of the same subsets as the training data, in proportion that matches the training sampling frequencies. During evaluation the retrieval database consists in the full union of these datasets, with the exception of books for which we use a sub-sample of 4%. The evaluation retrieval database thus contains 1.75T tokens. To limit test set leakage, we compute the 13-gram Jaccard similarity between train and test documents using the MinHash scheme and remove all training documents with high similarity (0.8 or higher) to a validation or test set document. Additionally, we remove all validation and test articles from Wikitext103 (Merity et al., 2017) from our Wikipedia training data.

## 2.2. Retrieval-enhanced autoregressive token models

Our approach uses retrieval as a way to augment input examples at the granularity of small chunks of tokens. Formally, we consider sequences of integer tokens in  $\mathbb{V} = [1, v]$ , obtained using a text tokenizer<sup>1</sup>. We split each  $n$ -token-long example  $X = (x_1, \dots, x_n)$  into a sequence of  $l$  chunks  $(C_1, \dots, C_l)$  of size  $m = \frac{n}{l}$ , i.e.  $C_1 \triangleq (x_1, \dots, x_m), \dots, C_l \triangleq (x_{n-m+1}, \dots, x_n) \in \mathbb{V}^m$ . We use  $n = 2048$  and  $m = 64$ . We augment each chunk  $C_u$  with a set  $\text{RET}_{\mathcal{D}}(C_u)$  of  $k$  neighbours from the database  $\mathcal{D}$ .  $\text{RET}_{\mathcal{D}}$  (or

<sup>1</sup>We use the notation  $[1, v] \triangleq \{1, \dots, v\}$  throughout the text.

$\text{RET}$  for brevity) is a non-trainable operator specified in §2.3. Token likelihoods are provided by a model, parameterized by  $\theta$ , that takes as input both previous tokens and their retrieved neighbours. This defines the following retrieval-enhanced sequence log-likelihood:

$$L(X|\theta, \mathcal{D}) \triangleq \sum_{u=1}^l \sum_{i=1}^m \ell_\theta(x_{(u-1)m+i} | (x_j)_{j < (u-1)m+i}, (\text{RET}_{\mathcal{D}}(C_{u'}))_{u' < u}). \quad (1)$$

We set  $\text{RET}(C_1) = \emptyset$ , namely the likelihood of tokens from the first chunk does not depend on any retrieval data. This likelihood definition preserves *autoregressivity*: the probability of the  $i$ -th token of the  $u$ -th chunk,  $x_{(u-1)m+i}$ , only depends on previously seen tokens  $(x_j)_{1 \leq j < (u-1)m+i}$  and on the data retrieved from the previous chunks  $(\text{RET}(C_{u'}))_{u' < u}$ . We can therefore directly *sample* with log-probability  $\ell$ , where sampling within the chunk  $C_u$  is conditioned on the neighbours  $(\text{RET}(C_{u'}))_{u' < u}$ . This makes retrieval-enhanced models directly comparable with the largest language models that are evaluated by sampling.

### 2.3. Nearest neighbour retrieval

**Retrieval neighbours.** Our database consists of a key-value memory. Each value consists of two contiguous chunks of tokens which we denote  $[N, F]$  where  $N$  is the *neighbour* chunk which is used to compute the key, and  $F$  is its *continuation* in the original document. The corresponding key is the BERT embedding of  $N$ , averaged over time, that we denote  $\text{BERT}(N)$ . For each chunk  $C$ , we retrieve its approximate  $k$ -nearest neighbours from our key-value database using the  $L_2$  distance on BERT embeddings  $d(C, N) = \|\text{BERT}(C) - \text{BERT}(N)\|_2^2$ . The model receives the corresponding values  $\text{RET}(C) \triangleq ([N^1, F^1], \dots, [N^k, F^k])$ . Both neighbour chunks and their continuations provide meaningful improvements, as illustrated in our ablation study (Appendix D). We use a length 64 for both  $N^j$  and  $F^j$ , thus  $\text{RET}(C)$  has a shape of  $k \times r$  with  $r = 128$ . To avoid retrieving the chunk  $C_{u+1}$  in the retrieval set  $\text{RET}(C_u)$ , which would break causality during training, we filter out neighbours originating from the same document as the training sequence  $X$ .

For a database of  $T$  elements, we can query the approximate nearest neighbours in  $\mathcal{O}(\log T)$  time. We use the SCaNN library (Guo et al., 2020) to achieve this. This means that we can query our 2 trillion token database in 10 ms whilst evaluating or sampling from the model; this expense is amortized over a chunk length. Performing retrieval on-the-fly is too slow to keep up with the training calculations—we leverage the frozen aspect of the embedding operator BERT to precompute all approximate nearest neighbours and save the results as part of the data. In Fig. 9 in the Appendix, we show results where we only retrieve neighbours within Wikipedia. We find that neighbours tend to come from 2-3 links away from a given article whereas random articles are more than 5 links apart.

Table 1 | **MassiveText**. The last column indicates the sampling weight during training. The multilingual subsets include documents in 10 languages. The full breakdown is given in §A.1.

Source	Token count (M)	Documents (M)	Multilingual	Sampling frequency
Web	977,563	1,208	Yes	55%
Books	3,423,740	20	No	25%
News	236,918	398	No	10%
Wikipedia	13,288	23	Yes	5%
GitHub	374,952	143	No	5%

## 2.4. RETRO model architecture

Our model relies on an encoder-decoder transformer architecture, integrating the retrieved data through a cross-attention mechanism as introduced in [Vaswani et al. \(2017\)](#). First, the retrieved tokens  $\text{RET}(C)$  are fed into an encoder Transformer, which computes the encoded neighbours set  $E$ . Denoting the intermediate activations by  $H$ , our transformer decoder then interleaves RETRO-blocks  $\text{RETRO}(H, E)$  and standard Transformer blocks  $\text{LM}(H)$  (the hyperparameter  $P \subseteq [1, L]$  determines at which layers we use a RETRO-block). These blocks are built from three different residual operators with signature  $\mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times d}$ : a fully-connected layer  $\text{FFW}$ , the standard sequence-level self-attention layer  $\text{ATTN}$ , and a chunked cross-attention layer  $\text{CCA}(\cdot, E)$  that incorporates information from the retrieval encoder:

$$\text{RETRO}(H, E) \triangleq \text{FFW}(\text{CCA}(\text{ATTN}(H), E)), \quad \text{and} \quad \text{LM}(H) \triangleq \text{FFW}(\text{ATTN}(H)) \quad (2)$$

Since  $\text{FFW}$ ,  $\text{ATTN}$  and  $\text{CCA}$  are all autoregressive operators whose output at position  $i$  only depends on  $(h_j)_{j \leq i}$ , any succession of RETRO and LM layers, followed by a token classification head defines an autoregressive log-likelihood (1). An overview of the model architecture is given in [Algorithm 1](#) and in [Fig. 2](#). We next describe the retrieval encoder and the chunked cross-attention layer in more detail, and explain how to sample from RETRO.

**Encoding retrieval neighbours.** For each chunk  $C_u$ , the  $k$  retrieval neighbours  $\text{RET}(C_u)$  are fed into a bi-directional transformer ENCODER, yielding the outputs  $E_u^j \triangleq \text{ENCODER}(\text{RET}(C_u)^j, H_u) \in \mathbb{R}^{r \times d'}$ , where  $j \in [1, k]$  indexes each neighbour. The retrieval encoder is a non-causal transformer. It is conditioned on  $H_u$ , the activations of chunk  $C_u$ , through cross-attention layers; this allows the representations of the retrieval encoder to be modulated by the retrieving chunk in a differentiable way. More precisely, the encoding of the  $j^{\text{th}}$  neighbour of the  $u^{\text{th}}$  chunk,  $\text{RET}(C_u)^j$ , depends on the attended activation  $H_u \triangleq (h_{(u-1)m+i})_{i \in [1, m]} \in \mathbb{R}^{m \times d}$  of chunk  $C_u$  at layer  $\min(P)$ . All neighbours for all chunks are encoded in parallel, yielding a full encoded set  $E \triangleq (E_u^j)_{u \in [1, l], j \in [1, k]} \in \mathbb{R}^{l \times k \times r \times d'}$ . We denote  $E_u \in \mathbb{R}^{k \times r \times d'}$  as the encoded neighbours for chunk  $u \in [1, l]$ .

**Chunked cross-attention.** To perform the CCA operation, we first split a given intermediate activation  $H \in \mathbb{R}^{n \times d}$  into  $l-1$  attending chunks  $\left(H_u^+ \triangleq (h_{u(m+i-1)})_{i \in [1, m]} \in \mathbb{R}^{m \times d}\right)_{u \in [1, l-1]}$ , as depicted on the right of [Fig. 2](#).  $H_u^+$  holds the intermediary embeddings of the last token in chunk  $C_u$  and of the first  $m-1$  tokens in  $C_{u+1}$ <sup>2</sup>. We compute the cross-attention between  $H_u^+$  and  $E_u$ —the encoded retrieval set obtained from chunk  $C_u$ . Attention is computed across time and across neighbours simultaneously, as we merge the neighbour and time dimensions of  $E_u$  before applying cross-attention. Since there is a notion of alignment between data chunks and retrieval neighbours, we use relative positional encodings as described in [§B.1.2](#).

We concatenate the  $l-1$  outputs of the per-chunk cross-attentions (each of shape  $m \times d$ ) across time, and properly pad the result; we thus form the output activation  $\text{CCA}(H, E) \in \mathbb{R}^{n \times d}$ . Formally, for each chunk  $C_u$  and for each token  $i \in [1, m]$  we set

$$\text{CCA}(H, E)_{u(m+i-1)} \triangleq \text{CA}(h_{u(m+i-1)}, E_u), \quad (3)$$

<sup>2</sup>The last token of chunk  $C_u$  is the first to be able to access the retrieved content  $E_u$  while maintaining autoregressivity in (1). Hence, there is a one token overlap between chunk  $C_u = (x_{(u-1)m+i})_{i \in [1, m]}$  and the corresponding attending chunk  $C_u^+ \triangleq (x_{u(m+i-1)})_{i \in [1, m]}$ .

Algorithm 1: Overview of RETRO model architecture.

**Hyperparam:**  $P$  and  $P_{\text{enc}}$ , indices of layers with cross-attention in the decoder and encoder respectively

**Hyperparam:**  $L$  and  $L_{\text{enc}}$ , number of decoder layers and number of encoder layers.

**Input:**  $X \in \mathbb{V}^n$ : sequence of tokens.  $(\text{RET}(C_u))_{1 \leq u \leq l}$ : the retrieved neighbours

**Output:**  $O \in \mathbb{R}^{n \times |\mathbb{V}|}$ : the output logits

```

def ENCODER( $\text{RET}(C_u)_{1 \leq u \leq l}, H$ ):
     $(H_u)_{u \in [1, l]} \leftarrow \text{SPLIT}(H)$ 
    for  $j \in [1, k], u \in [1, l]$  do // Encoder shared across neighbours and chunks
         $E_u^j = \text{EMB}_{\text{enc}}(\text{RET}(C_u)^j)$  // May be shared with the decoder EMB
        for  $p' \in [1, L_{\text{enc}}]$  do
             $E_u^j \leftarrow \text{ATTN}_{\text{enc}}(E_u^j)$  // Bi-directional attention
            if  $p' \in P_{\text{enc}}$  then
                 $E_u^j \leftarrow \text{CA}_{\text{enc}}(E_u^j, H_u)$ 
             $E_u^j \leftarrow \text{FFW}_{\text{enc}}(E_u^j)$ 
    return  $E$ 

 $H \leftarrow \text{EMB}(X)$ 
for  $p \in [1, L]$  do
     $H \leftarrow \text{ATTN}(H)$  // Causal attention
    if  $p = \min(P)$  then
        // The neighbour ENCODER is conditioned with the decoder activations of
        // the last layer before the first cross-attention
         $E = \text{ENCODER}(\text{RET}(C_u)_{1 \leq u \leq l}, H)$ 
    if  $p \in P$  then
         $H \leftarrow \text{CCA}(H, E)$ 
     $H \leftarrow \text{FFW}(H)$ 
 $O \leftarrow \text{READ}(H)$ 

```

where  $\text{CA}$  is the cross-attention residual operator over time-concatenated encoded neighbours. We recall that this operator is defined in its simplest version by three parameter matrices  $K \in \mathbb{R}^{d \times c}$ ,  $Q \in \mathbb{R}^{d \times c}$  and  $V \in \mathbb{R}^{d \times d}$ . For all  $h \in \mathbb{R}^d$  and  $Y \in \mathbb{R}^{T \times d}$ , we define

$$\text{CA}(h, Y) \triangleq \text{softmax}(YKQ^Th)YV, \quad (4)$$

where the softmax is performed on the second dimension and all products are matrix products. We use multi-head cross-attention, and add positional encodings to the softmax (see §B.1.2).

The first  $m - 1$  tokens cannot attend to any neighbour of a previous chunk; at these positions, we define  $\text{CCA}$  as the identity, setting  $\text{CCA}(H, E)_j \triangleq h_j$  for all tokens  $j \in [1, m - 1]$ . Finally, the last token  $h_{lm}$  attends to the last retrieval set  $E_l$  and we set  $h_{lm} \triangleq \text{CA}(h_{lm}, E_l)$  (not shown in Fig. 2). Listing 1 contains a simplified implementation of  $\text{CCA}$ . Note that chunked cross-attention is autoregressive: the output of  $\text{CCA}$  at position  $i$  depends on the sequence from tokens from 0 to  $i$  that is input to  $\text{CCA}$ .

With RETRO models, even though each  $\text{CCA}$  cross-attention attends only to the neighbours of the preceding chunk  $\text{RET}(C_{u-1})$ , the dependencies over previous neighbours are propagated via the self-attention operations. The activations of the  $i^{\text{th}}$  token in the  $u^{\text{th}}$  chunk therefore potentially depend upon the set of *all* previous neighbours  $\text{RET}(C_{u'})_{u' < u}$ , without incurring the quadratic cost of cross attending to that set.

**Sampling.** When sampling, at the end of a chunk  $C_u$ , we use SCaNN to retrieve neighbours  $\text{RET}(C_u)$ , based on the embedding  $\text{BERT}(C_u)$ . The encoded neighbours  $E_u = \text{ENCODER}(\text{RET}(C_u))$  are then used to condition the generation of the next chunk  $C_{u+1}$ , which we do incrementally: overall the cost of sampling is thus quadratic in the size of the sampled sequence, as when sampling from regular Transformers; the added cost of retrieval is linear in the number of chunks  $l$ , and is negligible compared to the token sampling cost in practice.

## 2.5. Baseline Transformer architecture

We use a transformer (Vaswani et al., 2017) similar to the one described in (Radford et al., 2019), with some minimal changes: we replace LayerNorm with RMSNorm (Zhang and Sennrich, 2019) and use relative position encodings (Dai et al., 2019). As baselines, we train retrieval-free transformers with 132M, 368M, 1.3B and 7.0B parameters (embedding matrices are excluded from parameter counts). The hyperparameters we used are detailed in Table 2. All retrieval models use the same size encoder for the retrieval data, with  $d' = 896$  and 2 layers, which roughly adds 19M parameters. The encoder uses relative positional encodings. The retrieval models contain one RETRO-block every 3 blocks, starting from layer 6. For our smallest model, CCA is applied in layers 6, 9 and 12 of the main pathway and also once for query conditioning in the encoder, which adds an additional 12M parameters. The relative number of extra parameters reduces as we increase the baseline model size. All models are implemented using JAX (Bradbury et al., 2018) and Haiku (Hennigan et al., 2020).

## 2.6. Quantifying dataset leakage exploitation

RETRO models may arguably benefit more easily from evaluation dataset leakage, i.e. the fact that we evaluate on data that were also present in the training set. To better understand how retrieval improves language modelling performance, we therefore quantify evaluation likelihood as a function of the overlap between the evaluation and training datasets.

The following approach can be used with any language model, and depends only on the frozen retriever system presented in §2.3. We split the evaluation sequences  $(X_i)_i$  into chunks of length  $m \leq 64$ , and we see the training data as a set of chunks  $C$ . For each evaluation chunk  $C \in C$ , we retrieve the 10 closest neighbours (of length up to 128) in the training data. We then compute the longest token substring common to both the evaluation chunk and its neighbours. This gives a number  $s \in [0, m]$ . The value  $r(C) = \frac{s}{m}$ , ranging from 0 (chunk never seen) to 1 (chunk entirely seen), gives a reliable indication of how much overlap there is between the evaluation chunk and the training data. For a given model, we then obtain the log-likelihood  $\ell(C)$  of each chunk  $C$ , and the number of bytes  $N(C)$  it encodes. We then consider the filtered bits-per-bytes of the model:

$$\forall \alpha \in [0, 1], \quad C_\alpha \triangleq \{C \in C, r(C) \leq \alpha\}, \quad \text{bpb}(\alpha) \triangleq \frac{\sum_{C \in C_\alpha} \ell(C)}{\sum_{C \in C_\alpha} N(C)}, \quad (5)$$

Table 2 | **Number of parameters** for our baseline and RETRO models, excluding embeddings, along with the corresponding hyperparameters.

Baseline parameters	RETRO	$d$	$d_{\text{ffw}}$	# heads	Head size	# layers
132M	172M (+30%)	896	3,584	16	64	12
368M	425M (+15%)	1,536	6,144	12	128	12
1,309M	1,451M (+11%)	2,048	8,192	16	128	24
6,982M	7,532M (+8%)	4,096	16,384	32	128	32

which correspond to the bits-per-bytes on the set of chunks that overlap less than  $\alpha\%$  with the training chunks. Note that the full evaluation bit-per-bytes performance is recovered by  $bpb(1)$ . The function  $bpb(\cdot)$  allows us to evaluate the impact of evaluation leakage over predictive performance: for low  $\alpha$ ,  $bpb(\alpha)$  gives an indication on how the model performs on chunks that are entirely new; the slope of  $bpb(\cdot)$  shows how much the model exploits evaluation leakage.

### 3. Related Work

We first review existing work on using retrieval for language modelling, and compare **RETRO** to these works (see Table 3). As we train **RETRO** models on a large dataset containing a substantial section of the internet, our work raises potential privacy, safety, and fairness issues that we then review.

#### 3.1. Retrieval for language modelling

[Brants et al. \(2007\)](#) show that scaling the training data to trillions of tokens improves the machine translation performance of  $n$ -gram models. More recently, GPT-2 ([Radford et al., 2019](#)), GPT-3 ([Brown et al., 2020](#)), and Jurassic-1 ([Lieber et al., 2021](#)) show that scaling up language models leads to massive improvements on many downstream tasks. At the same time, [Carlini et al. \(2021\)](#) demonstrate that large-scale language models can perfectly memorise parts of their training data, suggesting that enhancing models with retrieval may lead to further improvements. However, significant leakage between train and test datasets ([Lee et al., 2021](#); [Lewis et al., 2021](#)) makes comparing and evaluating large models trained on large datasets difficult, especially once retrieval capabilities over the training dataset are added.

Historically, information retrieval for text relies on inverted index matching such as TF-IDF and BM25 ([Robertson and Zaragoza, 2009](#)). Foundational work use latent topic modelling approaches like LDA ([Blei et al., 2003](#)) to identify relevant neighbours ([Wei and Croft, 2006](#)). Work in machine translation such as [Zhang et al. \(2018\)](#) and [Gu et al. \(2018\)](#) retrieve translation pairs based on edit distance between source sentences and guide the translation output using the closest retrieved target sentences. The retrieval database may also be structured — for example, [Ahn et al. \(2016\)](#) use a symbolic knowledge graph to improve an RNN language model.

With the success of deep learning, retrieving systems have partly switched to dense learned representations based on a neural network’s activations. Continuous cache ([Grave et al., 2017](#)) adds probability mass to tokens for which previous activations resemble the current activation vector, extending the model’s context to the local history.  $k$ NN-LM ([Khandelwal et al., 2020](#)) applies this idea to transformers and extends the retrieval database to English Wikipedia, resulting in

Table 3 | Comparison of **RETRO** with existing retrieval approaches.

	# Retrieval tokens	Granularity	Retriever training	Retrieval integration
Continuous Cache	$O(10^3)$	Token	Frozen (LSTM)	Add to probs
$k$ NN-LM	$O(10^9)$	Token	Frozen (Transformer)	Add to probs
SPALM	$O(10^9)$	Token	Frozen (Transformer)	Gated logits
DPR	$O(10^9)$	Prompt	Contrastive proxy	Extractive QA
REALM	$O(10^9)$	Prompt	End-to-End	Prepend to prompt
RAG	$O(10^9)$	Prompt	Fine-tuned DPR	Cross-attention
FID	$O(10^9)$	Prompt	Frozen DPR	Cross-attention
EMDR <sup>2</sup>	$O(10^9)$	Prompt	End-to-End (EM)	Cross-attention
<b>RETRO (ours)</b>	$O(10^{12})$	Chunk	<b>Frozen (BERT)</b>	<b>Chunked cross-attention</b>

substantial improvements on Wikitext103 evaluation. Continuous cache and  $k$ NN-LM do not modify the underlying neural-network models, but interpolate at inference between the language model’s output and distributions computed from retrieved tokens. These methods can therefore be plugged into any model without additional training, although this limits the model’s ability to reason about the retrieved text. SPALM (Yogatama et al., 2021) addresses this limitation by adding an extra gating network to post-process the retrieved data; yet most of the network is unaffected by the retrieval during inference.

The retrieval representations may be trained directly instead of relying on a pre-trained model—retriever systems have been developed for this purpose, primarily on open-domain question answering. For example, DPR (Karpukhin et al., 2020) trains two BERT models (for queries and keys respectively) using a contrastive loss to align the representations of a question and of its answers. Lee et al. (2019) use an inverse cloze task to find semantic representations of passages for retrieval. These works differs from continuous cache and  $k$ NN-LM in that they embeds passages (or chunks) of text together, as opposed to each token individually. The retriever network is trained in isolation of the downstream task that uses the retrieval data. This potential issue is specifically addressed by REALM (Guu et al., 2020), which trains the retrieval system end-to-end to maximize the final training cross-entropy. This comes with the extra complexity of searching the database during training and periodically updating the embedding table, severely limiting the scale at which it can operate. RAG (Lewis et al., 2020) and FID (Izacard and Grave, 2021) build upon DPR to set the state of the art on question answering benchmarks by training encoder-decoder transformer models. More recently, EMDR<sup>2</sup> (Sachan et al., 2021) extends FID by using an expectation-maximization algorithm to train the retriever end-to-end and achieves state of the art results compared to similarly sized models.

In the open-domain dialogue setting, BlenderBot 2.0 (Komeili et al., 2021) learns to issue textual internet queries, outperforming dense retrieval methods when evaluated on a task measuring how close model responses are to those of humans. This involves collecting a dataset of human dialogues with associated search queries, which limits the scalability of this approach. Hashemi et al. (2020) introduce the Guided Transformer, a modified Transformer similar to RETRO, for document retrieval and clarifying question selection. Although effective on question answering and other tasks with strong conditioning, none of these methods are designed to model arbitrary text sequences, in contrast with RETRO.

RETRO shares components with  $k$ NN-LM and DPR in that it uses frozen retrieval representations. RETRO models longer sequences than QA examples; this requires to reason at a sub-sequence level, and to retrieve different documents for the different chunks of a sequence. Similar to FID, RETRO processes the retrieved neighbours separately in the encoder, and assemble them in the chunked cross-attention. This differs from e.g. REALM, that prepends retrieved documents to the prompt. Using chunks allows for repeated retrieval whilst generating a sequence as opposed to retrieving only once based on the prompt alone. Furthermore, retrieval is done during the whole pre-training process in RETRO, and is not simply plugged-in to solve a certain downstream task. Finally, previous methods based on dense query vectors use small models and retrieval datasets with less than 3B tokens (English Wikipedia). Table 3 summarizes the difference of RETRO with existing approaches.

### 3.2. Privacy, safety and fairness

Bender et al. (2021); Weidinger et al. (2021) highlight several dangers of large language models. Those stem from their ability to memorise training data, their high training cost, the static nature of their training data (Lazaridou et al., 2021), their tendency of amplifying inherent biases in the training data, and their ability to generate toxic language (Gehman et al., 2020). In this section we inspect these dangers, focusing on how retrieval augmented language models may exacerbate or

mitigate them.

Large language models can perfectly memorise parts of their training data (Carlini et al., 2021). When coupled with large training datasets gathered from the web or other sources, this has clear privacy and safety implications. Retrieval models such as RETRO that have access to the entire training dataset during inference exacerbate these privacy issues by being able to directly copy training data. However, retrieval systems offer a path towards mitigating these concerns via obliteration of the retrievable data at inference time. In addition, differential privacy training (Abadi et al., 2016) of retrieval models could guarantee that no private information is stored in the model weights, while individualisation on private data could be made by updating the retrieval database at inference time.

Due to their high training cost, re-training large language model regularly to incorporate new data, languages, and norms is prohibitively expensive. To keep retrieval models up-to-date, it may be sufficient to update the retrieval database, which is orders of magnitude cheaper than re-training a model from scratch. In addition to the benefits of updating models in terms of fairness and bias, simply training large language models has a significant energy cost (Schwartz et al., 2020; Strubell et al., 2019). Retrieval mechanisms offer a path to reducing the compute requirements needed to train and update language models that reach a certain performance.

Large language models are prone to generating toxic outputs, as shown in Gehman et al. (2020). Bender et al. (2021); Jo and Gebru (2020) advocate for the importance of better training data curation and documentation. Additionally, if portions of the training data are found to be eliciting biased or toxic outputs after training, retrieval allows for some correction, as the offending retrieval data can be retroactively filtered. However, it is also the case that without careful analysis and intervention, retrieval models may exacerbate biases that are present in the training data. Retrieval models can also add a further source of bias through the selection mechanism for retrieval documents. Further work in this area is required to better understand how retrieval affects the bias and toxicity of the model outputs.

Finally, samples from large models are difficult to interpret, making mitigating these issues all the more challenging (Belinkov et al., 2020; Jain and Wallace, 2019). Retrieval provides more insights into the outputs of a model, as one can directly visualise or modify the neighbours that are being used. The examples in Table 6, 7, 20 and 21 illustrate how retrieval makes language models more factual and interpretable by providing more transparent outputs.

## 4. Results

We first report results on language modelling benchmarks. Second, we show how to Retrofit pre-trained Transformer language models into retrieval models with few additional FLOPs. Next, we report RETRO results on question answering. Finally, we report evaluation metrics with leakage filtering, to better understand the source of the gains with retrieval.

### 4.1. Language modelling

**Datasets.** We evaluate our models on C4 (Raffel et al., 2020), WikiText103 (Merity et al., 2017), Curation Corpus (Curation, 2020), Lambada (Paperno et al., 2016) and the Pile (Gao et al., 2020). We also evaluate on a set of manually selected Wikipedia articles that were added or heavily edited in September 2021, months after our pre-training and retrieval dataset was collected (details are given in §A.2). We construct the dataset with articles from the “future” and manually remove new articles that strongly overlap documents in our training data. This guarantees that the evaluation documents are not leaked in our training data.

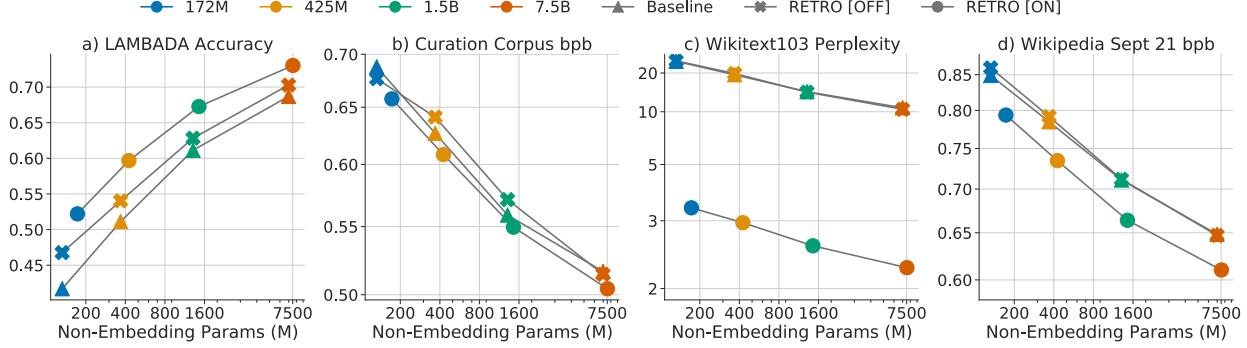


Figure 3 | **Scaling with respect to model size.** (a) LAMBADA top-1 accuracy. (b) Evaluation loss on curation corpus. (c) Perplexity on Wikitext103 valid. (d) Bits-per-byte on selected Wikipedia articles from September 2021.

For C4, Wikitext103, the Pile, and our Wikipedia dataset we evaluate the language modelling performance on entire documents and measure the bits-per-byte (bpb). We favour bits-per-byte over loss as it is tokenizer agnostic. We evaluate with a sequence length of 2048 tokens but use a stride of 1024 within documents to mitigate boundary effects. On Curation Corpus we concatenate the article, the “TL;DR :” string, and the summary, but only evaluate the bpb on the summary. For Lambada we evaluate the accuracy on the last word, using greedy generation.

**Model scaling.** In Fig. 1(left) and Fig. 3 we show the language modelling performance as we scale models from 150 million to 7 billion (non-embedding) parameters. We see that on all datasets, RETRO outperforms the baseline at all model sizes. Furthermore, we observe that improvements do not diminish as we scale the models. The performance is dataset dependent, with the largest gains on Wikitext103 and C4. Wikipedia articles and other web pages are similar to Wikitext103 documents, even if not exact copies (§4.4), we thus obtain dramatic improvements on Wikitext103 as our retrieval model is able to directly exploit these overlaps. The smallest gains are for Curation Corpus, where RETRO only slightly outperforms the baseline. This is expected as Curation Corpus summaries are designed to only contain information from the source article and are not included in our retrieval database. On our “future” Wikipedia September 2021 dataset, we also observe consistent gains for all model sizes.

**Data scaling.** Fig. 1 (middle) shows how scaling the retrieval database at evaluation improves the language modelling performance. We observe dramatic gains as the retrieval data is increased from Wikipedia (4 billion tokens) to all of Massive text (1.7T tokens). Fig. 1(right) shows how performance scales as we increase the number of retrieved chunks. Despite being only trained with 2 neighbours, we see consistent improvements for all models when the number of neighbours is increased from 1 to 10. Furthermore, we observe that larger models are able to better utilise more neighbours: the 172M model improves with up to 10 neighbours, whereas the 7B model improves with up to 40 neighbours.

**The Pile.** We evaluate our 7B models on the Pile test sets<sup>3</sup> and compare against the 178B parameter Jurassic-1 (Lieber et al., 2021) model and the 280B parameter Gopher (Rae et al., 2021) model. We do not compare against GPT-3 as it is outperformed by Jurassic-1 and Gopher on almost all subsets. Fig. 4 shows the relative improvements in bits-per-byte over our 7B transformer baseline for our

<sup>3</sup>Due to legal and ethical concerns relating to their use, we exclude the Enron Emails and the Youtube Subtitles datasets.

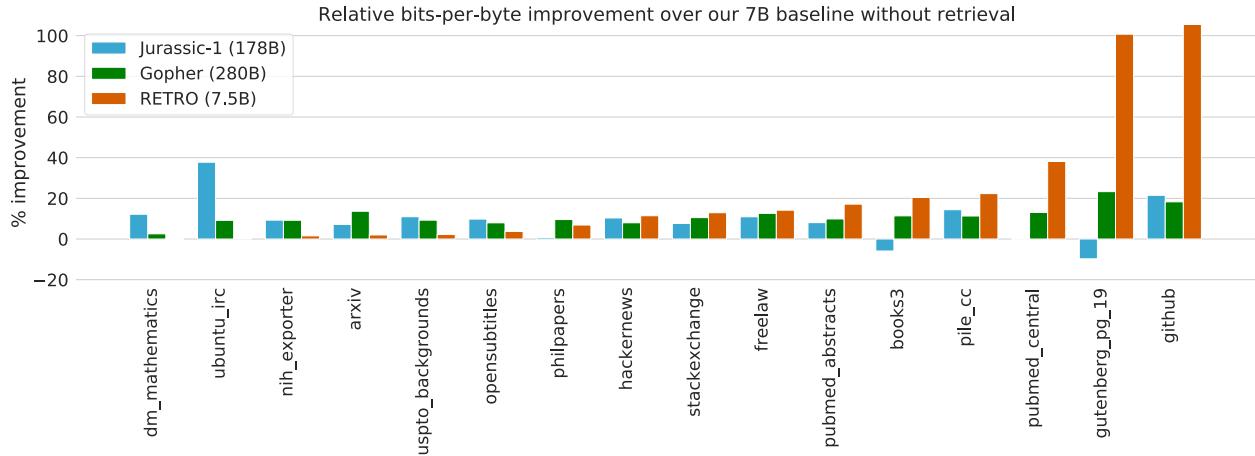


Figure 4 | **The Pile: Comparison of our 7B baseline against Jurassic-1, Gopher, and RETRO.** We observe that the retrieval model outperforms the baseline on all test sets and outperforms Jurassic-1 on a majority of them, despite being over an order of magnitude smaller.

7.5B RETRO model, Jurassic-1 and Gopher. Jurassic-1 outperforms the baseline on all datasets except for books, likely due to the inclusion of books in our training data. Gopher and RETRO outperform the baseline on all test sets. Overall, RETRO 7.5B outperforms Jurassic-1 and Gopher on a majority of the test sets. On the `dm_mathematics` and `ubuntu_irc` subsets, our RETRO model does not outperform our 7B baseline and underperforms Jurassic-1. We hypothesise that the retrieved neighbours on these datasets are not helpful, due to a combination of what is in our retrieval dataset and the efficacy of the nearest-neighbour search.

**Wikitext103.** To validate our approach in a controlled setting, we compare our method with  $k$ NN-LM (Khandelwal et al., 2020) on the Wikitext103 dataset in Table 4. We train a baseline transformer on the training set of Wikitext103. This transformer has 24 layers, 1024 hidden units, 16 heads and a key size of 64, as in Baevski and Auli (2019). Our baseline does not have adaptive input, and our tokenizer has an open vocabulary, unlike Baevski and Auli (2019), which makes our baseline

**Table 4 | Perplexities on Wikitext103.** When using the Wikipedia dataset for retrieval, RETRO performs similarly to our implementation of  $k$ NN-LM. As we scale the retrieval dataset, RETRO performs much better. The perplexities for retrieving from full MassiveText are quite low, which is partly due to partial overlap with Wikitext103 not caught by our deduplication.

Model	Retrieval Set	#Database tokens	#Database keys	Valid	Test
Adaptive Inputs (Baevski and Auli, 2019)	-	-	-	17.96	18.65
SPALM (Yogatama et al., 2021)	Wikipedia	3B	3B	17.20	17.60
$k$ NN-LM (Khandelwal et al., 2020)	Wikipedia	3B	3B	16.06	16.12
Megatron (Shoeybi et al., 2019)	-	-	-	-	10.81
Baseline transformer (ours)	-	-	-	21.53	22.96
$k$ NN-LM (ours)	Wikipedia	4B	4B	18.52	19.54
RETRO	Wikipedia	4B	0.06B	18.46	18.97
RETRO	C4	174B	2.9B	12.87	10.23
RETRO	MassiveText (1%)	18B	0.8B	18.92	20.33
RETRO	MassiveText (10%)	179B	4B	13.54	14.95
RETRO	MassiveText (100%)	1792B	28B	<b>3.21</b>	<b>3.92</b>

perplexities a bit higher. The full experiment details and hyperparameters are given in §C.2 and Table 11.

We re-implement  $k$ NN-LM with our tokenizer and baseline transformer to produce embeddings of size 1024 for every token in Wikitext103.  $k$ NN-LM has probabilities  $p_{k\text{NN-LM}} = \lambda p_{k\text{NN}} + (1 - \lambda)p_{\text{LM}}$  with  $p_{k\text{NN}}(n_k) \propto \exp(-\alpha d_k)$ . We tune  $\lambda = 0.118$  and  $\alpha = 0.00785$  on the validation set (Fig. 7) and report performance for these hyperparameters on both the validation and test set.

We fine-tune our baseline transformer into a RETRO model (Fig. 7), using the Wikitext103 training data and retrieving from Wikipedia with 2 neighbours. We only train the new weights, as explained in §4.2, and share the embedding weights between the encoder and the main pathway. This is necessary for Wikitext103 which is quite small, as training RETRO from scratch in this setting leads to over-fitting.

We evaluate the fine-tuned RETRO model with different retrieval sets. We use 10 neighbours at evaluation for both RETRO and  $k$ NN-LM. When retrieving from Wikipedia, we obtain results comparable to our  $k$ NN-LM implementation. Furthermore, scaling the retrieval database to MassiveText yields dramatic improvements, though this is partly due to leakage (see §4.4). For reproducibility, we also include results when retrieving from C4, which are close to previous state-of-the-art and comparable to using 10 % of MassiveText.

It is worth noting that  $k$ NN-LM requires 1024 floats for every token in the retrieval dataset, totalling 15 terabytes (Tb) for the 4 billion tokens in Wikipedia.  $k$ NN-LM and other token-level retrieval approaches therefore don't scale to retrieval databases with trillions of tokens such as MassiveText. In comparison, RETRO only requires 215Gb to index our Wikipedia dataset, and 93Tb for MassiveText. Inspecting the number of retrieval database entries in Table 4 makes it clear why retrieving at the chunk level is necessary when scaling to datasets with trillions of tokens.

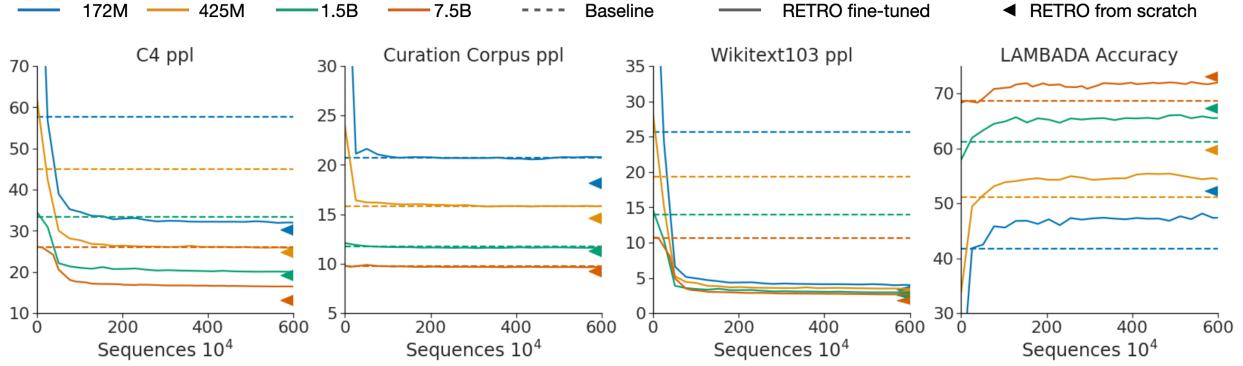
## 4.2. RETRO-fitting baseline models

We extend baseline models into RETRO models by freezing the pre-trained weights and training only chunked cross-attention and neighbour encoder parameters (less than 10% of weights for the 7B model) in Fig. 5. This offers an efficient alternative path to enhance transformers with retrieval, requiring only 6 million sequences (3% of the pre-training sequences that we used). Additionally, by only training the new weights we ensure that when evaluated without retrieval, the original model performance is exactly maintained. RETROfitting models quickly surpasses the performance of baseline models and even achieves performance close to that of RETRO models trained from scratch. The experiment hyperparameters are given in §C.3.

## 4.3. Question answering

We fine-tune our retrieval models on the Natural Questions (Kwiatkowski et al., 2019) dataset to demonstrate that our retrieval pathway can be used to inject information from arbitrary data sources. We use the version<sup>4</sup> provided by Izacard and Grave (2021) which is augmented with the retrieved passages from DPR (Karpukhin et al., 2020). We fine-tune all the weights of our 7.5B pre-trained RETRO model for 25,000 steps using the top 20 retrieved passages. We format the data as “question: {question} \nanswer: {answer}” and left pad the data such that “answer:” coincides with the end of the first chunk of 64 tokens and thus aligns with the first retrieving chunk. The model has access to the question via the previous tokens in the sequence as well as the top 20 DPR Wikipedia passages and their titles via the chunked cross-attention mechanism.

<sup>4</sup><https://github.com/facebookresearch/FiD>



**Figure 5 | RETRO-fitting a baseline transformer.** Any transformer can be fine-tuned into a retrieval-enhanced transformer by randomly initializing and training only the chunked cross-attention and retrieval encoder weights. Fine-tuning in this way quickly recovers and surpasses the non-retrieval performance, and almost achieves the same performance as training a retrieval model from scratch (shown by the arrow on the right hand side of each plot). We find good performance RETRO-fitting our models training on only 3% the number of tokens seen during pre-training.

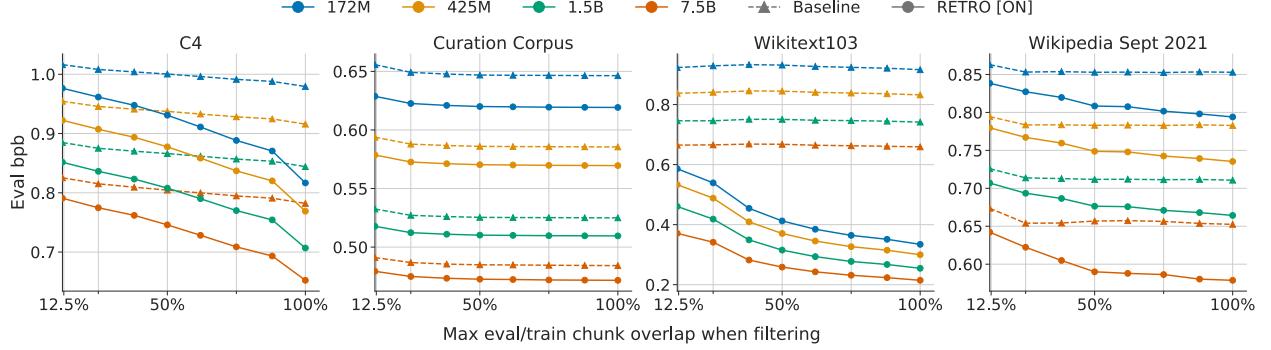
The exact match scores are shown in [Table 5](#) and the full fine-tuning details are given in [§C.4](#). Our method is competitive with previous approaches such as REALM, RAG and DPR, but underperforms the more recent F1D. In contrast with this work, we find that increasing the number of neighbours past 20 does not improve RETRO performance on this task. We hypothesise that the encoder-decoder structure of T5—the base model in F1D—and the T5 pre-training objective leads to a model that relies more on the encoder output than RETRO, which is important in the QA setting. To compete with T5-finetuned models, future work should consider ways of forcing RETRO to rely further on the retrieval encoder output when producing tokens.

#### 4.4. Relating retrieval performance to dataset leakage.

We report the filtered eval losses as detailed in [§2.6](#) on C4, Curation Corpus and Wikitext103 in [Fig. 6](#). On C4 and Wikitext103, for which there is leakage into the training set, the slope is negative for both baseline models and RETRO models. RETRO models exploit leakage more strongly than baseline models, as indicated by the more negative slope. This is due to its explicit ability to copy-paste existing training chunks to predict leaked evaluation chunks (see a qualitative example of this model behavior

**Table 5 | Question answering results.** Exact match accuracy on Natural Questions.

Model	Test Accuracy
REALM ( <a href="#">Guu et al., 2020</a> )	40.4
DPR ( <a href="#">Karpukhin et al., 2020</a> )	41.5
RAG ( <a href="#">Lewis et al., 2020</a> )	44.5
EMDR <sup>2</sup> ( <a href="#">Sachan et al., 2021</a> )	52.5
F1D ( <a href="#">Izacard and Grave, 2021</a> )	51.4
F1D + Distill. ( <a href="#">Izacard et al., 2020</a> )	54.7
Baseline 7B (closed book)	30.4
RETRO 7.5B (DPR retrieval)	45.5



**Figure 6 | Performance vs. longest common retrieval substring.** Evaluation loss as a function of allowed longest common substring between evaluation data chunks and their nearest neighbours. Retrieval still helps when considering chunks with no more than 8 contiguous tokens overlapping with training dataset chunks.

on a WikiText103 article in [Table 19](#)). On Curation Corpus, retrieval provides a constant offset, which is expected as there is by design no leakage between Curation Corpus and the training dataset.

On the other hand, RETRO outperforms baseline models at all leakage levels, down to  $\alpha = 12.5\%$ . At this level, the loss is computed on chunks with less than 8 contiguous tokens shared with the closest matching chunk in the training dataset—this is a reasonable level of overlap at which we consider that there is no local leakage. Retrieval thus improves predictions on both chunks that are syntactically similar to chunks in the training set, and on chunks that are syntactically different from all training chunks. This points toward a non trivial RETRO capacity of generalizing based on both model parameters and retrieval database. Similar results are found on the Pile dataset (see [Fig. 12](#), [§F.3](#)).

#### 4.5. Using RETRO for sampling

We show examples of samples obtained using the 7.5B RETRO model in [Table 6](#), [Table 7](#) and [Appendix E](#). For each chunk (the first one being the prompt), we juxtapose sampled chunks  $C_u$  with retrieved neighbours  $\text{RET}(C_u)$ . To give an indication of local overlap, we colour each sampled token in chunk  $C_u$  based on the length of the longest common prefix (LCP) found in the retrieved chunks  $\text{RET}(C_{u-1})$ . Similarly, we colour the retrieved chunks based on the LCP in the sampled chunk. For the sample in [Table 6](#), for which we chose the prompt, we observe that the retrieved chunks influence the sample as there are overlaps between the sampled tokens and neighbour tokens. Overall, retrieval reduces hallucinations (in line with the findings of [Shuster et al. \(2021\)](#)) and makes the model more knowledgeable, when comparing with samples produced with retrieval disabled. In the sample in [Table 7](#), the model recognises that the prompt is the beginning of the first scene of Hamlet and leverages retrieval data to continue it with only a few mistakes. We provide further examples in [Appendix E](#), including examples from the evaluation sets, as well as the detailed procedure used for colouring the tables.

## 5. Conclusion

We present Retrieval-Enhanced Transformers (RETRO), a method for modelling arbitrary text sequences whilst retrieving from databases with trillions of tokens—scaling the data available to models by an order of magnitude compared to what is typically consumed during training. RETRO models

gains do not diminish for models with up to at least 7B parameters, and correspond to non-retrieval models with 10 $\times$  more parameters on certain datasets. On Wikitext103 and the Pile, RETRO outperforms previous models trained on large scale datasets. We also show that RETRO is competitive on retrieval-intensive downstream tasks such as question answering.

RETRO models are flexible and can be used without retrieval at evaluation and still achieve comparable performance to baseline models. Conversely, baseline models can be rapidly fine-tuned into RETRO models to obtain nearly the same performance as if trained from scratch. Careful analysis shows that only a modest fraction of the gains obtained by RETRO are due to test set leakage. In general, we caution for such leakage in large-scale language datasets and suggest further work in better understanding the role of test set leakage in the performance of large-scale language models.

Overall, our work demonstrates at an unprecedented scale that semi-parametric approaches can provide an orthogonal, more efficient approach than raw parameter scaling as we seek to build more powerful language models.

## Acknowledgements

We would like to thank Nikolai Grigorev, Marc'aurelio Ranzato, Cyprien de Masson d'Autume, Po-Sen Huang, Johannes Welbl, Lisa Anne Hendricks, Ethan Perez, Jeff Stanway, Eric Noland, Gregory Wayne, John Jumper, Julian Schrittwieser, Lorrayne Bennett, Devang Agrawal, Dani Yogatama, Susannah Young, Nando de Freitas, Demis Hassabis, and Koray Kavukcuoglu for their help, advice and reviews. Additionally, we would like to thank Zonglin Li, David Simcha, and the ScaNN developers for their help.

**Table 6 | Sample - Beavers are interesting animals.** The RETRO [OFF] sample quickly diverges to other animals while the RETRO [ON] sample tends to stay focused on the beaver topic due to neighbour conditioning.

Prompt and sample of RETRO [OFF]	Prompt and sample of RETRO [ON] colored by LCP with RET( $C_u - 1$ ) LCP = 0, 1, 2, 3, 4, ≥ 5	[ $N_u^1, F_u^1$ ] colored by LCP with $C_{u+1}$ LCP = 0, 1, 2, 3, 4, ≥ 5	[ $N_u^2, F_u^2$ ] colored by LCP with $C_{u+1}$ LCP = 0, 1, 2, 3, 4, ≥ 5
Beavers are interesting animals that live near rivers. They build	Beavers are interesting animals that live near rivers. They build	.Beavers build their lodges in ponds they have created in wooded areas. Like many things in nature, there is a connection between creatures in the wild. Beaver ponds cause trees to drown, but the dead trees attract the great blue heron, which often return year after year. Over time, a beaver pond can attract more than 50 nests in a colony, called a rookery. An example of this can be found in the large pond off Bradford Road at Carter Fields near the Boxford Line. Chris Leahy, an expert with the Massachusetts Audubon Society who wrote	naw them into smaller sections and drag them into the water. Engineers of the Pond Beavers are interesting animals because they change the habitat in which they live. Beavers do this by blocking up streams to create ponds. Then they build their homes, called lodges, in these ponds. Beavers' bodies make them well-suited for underwater building. Special muscles close off their noses, ears, and throats to keep the water out. Beavers' broad tails act like rudders for steering. Their two very large, orange front teeth are used to gnaw down trees. They begin building their dam
dams to create ponds. Frogs are amphibians, so they can live in both land and water. They have great camouflage to hide from predators. The Golden Retriever, or Golden, is a very big dog. They are very strong, and have a thick coat to help them live in	their houses called beaver dams in the riverbeds. They also live on land. Beavers use their strong teeth and strong jaws to cut down trees and branches to build their homes. They also use their teeth and jaws to chew up food. Beavers use their big, flat tails to swim. They use	, then they mean that you are very busy. Beavers swim easily in streams, picking up rocks and sticks to build their dams. They gnaw at trees with their big front teeth to cut them down. Then they use parts of the trees to build their houses. Beavers are clever builders. They know exactly what they need to build their beaver dams. They use mud from the stream to make their dams stay together. They use their tails to pat down the mud. Beavers put a snug room at the top of their dams for their babies. They store their food underwater. Beavers eat the bark from the	ar-like tail, and two protruding teeth that are strong enough to gnaw down trees. The beaver uses trees, branches, and mud to build dams across rivers and streams. These dams create a deep pool of water in which the beaver builds its home. A beaver home is called a lodge. A baby beaver or "kit" remains in the family lodge until the age of two. Beaver fur, known as pelt, was once highly popular as a trim for hats and coats. How might the popularity of beaver fur contributed to the colonization of New York? www.Ww
cold areas. A kangaroo is an animal that is very large. It can jump very high. They can run very fast. A meerkat is a small animal that lives in Africa. They live in colonies of up to a hundred of them. They can climb trees very easily.	their sharp teeth to gnaw at rocks and cut trees. They use their flat tails to steer when they are swimming. In the winter, beavers have to build a house called a beaver lodge. They build a house that is very strong. The walls are made of twigs. The roof is made	, then they mean that you are very busy. Beavers swim easily in streams, picking up rocks and sticks to build their dams. They gnaw at trees with their big front teeth to cut them down. Then they use parts of the trees to build their houses. Beavers are clever builders. They know exactly what they need to build their beaver dams. They use mud from the stream to make their dams stay together. They use their tails to pat down the mud. Beavers put a snug room at the top of their dams for their babies. They store their food underwater. Beavers eat the bark from the	w at trees with their big front teeth to cut them down. Then they use parts of the trees to build their houses. Beavers are clever builders. They know exactly what they need to build their beaver dams. They use mud from the stream to make their dams stay together. They use their tails to pat down the mud. Beavers put a snug room at the top of their dams for their babies. They store their food underwater. Beavers eat the bark from the trees that they cut down! What is the main idea of the first paragraph? 2. What is the main idea of the second paragraph?
A mouse is a small mammal that lives on land. It is a very good climber and it can run very fast. Penguins are birds that live on Antarctica. They have a thick coat to keep them warm. Rabbits are small animals that live in the ground. They	of branches and other tree parts. They also use their strong jaws to cut trees. They bring them to their house. They also use their sharp teeth to chew up the tree parts. They use their flat tails to swim to the top of their house. Then they use their teeth and jaws to chew up the tree		

**Table 7 | Sample - Hamlet, Act 1, Scene 1.** The RETRO [OFF] sample has correct syntax but is hallucinated, and ends with repetition of one character (*FRANCISCO Approach me not*). The RETRO [ON] sample is the correct continuation of the original text, and is robust to formatting differences between our prompt and the retrieved data.

Prompt and sample of RETRO[OFF]	Prompt and sample of RETRO[ON] colored by LCP with $C_u-1$ LCP = <b>0, 1, 2, 3, 4, &gt; 5</b>	$[N_u^1, F_u^1]$ colored by LCP with $C_{u+1}$ LCP = <b>0, 1, 2, 3, 4, &gt; 5</b>	$[N_u^2, F_u^2]$ colored by LCP with $C_{u+1}$ LCP = <b>0, 1, 2, 3, 4, &gt; 5</b>
ACT I SCENE I. Elsinore. A platform before the castle. FRANCISCO at his post. Enter to him BERNARDO BERNARDO Who's there ? FRANCISCO Nay, answer me: stand, and unfold yourself. BERNARDO	ACT I SCENE I. Elsinore. A platform before the castle. FRANCISCO at his post. Enter to him BERNARDO BERNARDO Who's there ? FRANCISCO Nay, answer me: stand, and unfold yourself. BERNARDO	ACT I SCENE I. Elsinore. A platform before the castle. (FRANCISCO at his post. Enter to him BERNARDO) BERNARDO Who's there? FRANCISCO Nay, answer me: stand, and unfold yourself. BERNARDO Long live the king! FRANCISCO Bernardo? BERNARDO He. FRANCISCO You come most carefully upon your hour. BERNARDO 'Tis now struck twelve; get thee to bed, Francisco. FRANCISCO For this relief much thanks: 'tis bitter cold, And I am sick at heart. BERNARDO Have you	><TEXT>ACT I SCENE I. Elsinore. A platform before the castle. FRANCISCO at his post. Enter to him BERNARDO BERNARDO Who's there? FRANCISCO Nay, answer me: stand, and unfold yourself. BERNARDO Long live the king! FRANCISCO Bernardo? BERNARDO He. FRANCISCO You come most carefully upon your hour. BERNARDO 'Tis now struck twelve; get thee to bed, Francisco. FRANCISCO For this relief much thanks: 'tis bitter cold, And I am sick at heart.
Who calls ? FRANCISCO I am the lord here; I, Francisco, that am sick of grief. I Aside. BERNARDO The king ! FRANCISCO I am sick of that also. BERNARD O My lord ? FRANCISCO Do not approach me. BERNARDO	Long live the king! FRANCISCO Bernardo? BERNARDO He. FRANCISCO You come most carefully upon your hour. BERNARDO 'Tis now struck twelve; get thee to bed, Francisco. FRANCISCO For this relief much thanks: 'tis bitter cold, And I am sick at heart. B	Long live the king! FRANCISCO Bernardo? BERNARDO He. FRANCISCO You come most carefully upon your hour. BERNARDO 'Tis now struck twelve; get thee to bed, Francisco. FRANCISCO For this relief much thanks: 'tis bitter cold, And I am sick at heart. </TEXT></DOC><DOC>CNO>romeo</DOCNO><TEXT>ACT I PROLOGUE Two households, both alike in dignity, In fair Verona, where we lay our scene, From ancient grudge break to new mutiny,	live the king! FRANCISCO Bernardo? BERNARDO He. FRANCISCO You come most carefully upon your hour. BERNARDO 'Tis now struck twelve; get thee to bed, Francisco. FRANCISCO For this relief much thanks: 'tis bitter cold, And I am sick at heart. BERNARDO Have you had quiet guard? FRANCISCO Not a mouse stirring. BERNARDO Well, good night. If you do meet Horatio and Marcellus, The rivals of my watch, bid them make haste. FRANCISCO I think I hear them. Stand, ho! who is there? Enter
Francisco, I would speak with you. FRANCISCO Approach me not , but speak. BERNARDO Your hand, your voice FRANCISCO I will not hear thee speak. BERNARDO Francisco, your hand, I entre at thee. FRANCISCO Approach me not. BERNARDO Francisco FRANCISCO Approach me not. BERNARDO I have a letter FRANCISCO Approach me not. BERNARDO For the king. FRANCISCO Approach me not. BERNARDO There's no treason't. FRANCISCO Approach me not. BERNARDO I will	ERNARDO Have you had quiet guard? FRANCISCO Not a mouse stirring. BERNARDO Well, good night. If you do meet Horatio and Marcellus, The rivals of my watch, bid them make haste. FRANCISCO I think I hear them. Stand, ho! who is there? Enter	had quiet guard? FRANCISCO Not a mouse stirring. BERNARDO Well, good night. If you do meet Horatio and Marcellus, The rivals of my watch, bid them make haste. FRANCISCO I think I hear them. Stand, ho! who is there? (Enter HORATIO and MARCELLUS) HORATIO Friends to this ground. MARCELLUS And liegemen to the Dane. FRANCISCO Give you good night. MARCELLUS O, farewell, honest soldier: Who hath relieved you? FRANCISCO Bernardo has my place. Give you good night. (Exit)	ARDI Have you had quiet guard? FRANCISCO Not a mouse stirring. BERNARDO Well, good night. If you do meet Horatio and Marcellus, The rivals of my watch, bid them make haste. FRANCISCO I think I hear them.— Stand, ho! who is there? ENTER HORATIO AND MARCELLUS, HORATIO Friends to this ground. MARCELLUS And liegemen to the Dane. FRANCISCO Give you good night. MARCELLUS O, farewell, honest soldier: Who hath relieved you? FRANCISCO Bernardo hath my place. Give you good night

## A. Datasets

We provide a full description of MassiveText and of our extract of recent Wikipedia articles.

### A.1. Full description of MassiveText

The full break down of MassiveText by source and languages is given in [Table 8](#). For a full description and analysis of MassiveText, see [Rae et al. \(2021\)](#).

Source	Language	Token count (M)	Documents	Sampling weight
Web	En	483,002	604,938,816	0.314
	Ru	103,954	93,004,882	0.033
	Es	95,762	126,893,286	0.033
	Zh	95,152	121,813,451	0.033
	Fr	59,450	76,612,205	0.033
	De	57,546	77,242,640	0.033
	Pt	44,561	62,524,362	0.033
	It	35,255	42,565,093	0.033
	Sw	2,246	1,971,234	0.0044
	Ur	631	455,429	0.0011
Books	En	3,423,740	20,472,632	0.25
News	En	236,918	397,852,713	0.1
Wikipedia	En	3,977	6,267,214	0.0285
	De	2,155	3,307,818	0.003
	Fr	1,783	2,310,040	0.003
	Ru	1,411	2,767,039	0.003
	Es	1,270	2,885,013	0.003
	It	1,071	2,014,291	0.003
	Zh	927	1,654,772	0.003
	Pt	614	1,423,335	0.003
	Ur	61	344,811	0.0001
	Sw	15	58,090	0.0004
Github	-	374,952	142,881,832	0.05
Total	-	5,026,463	1,792,260,998	1

Table 8 | **MassiveText dataset.** The final column indicates the sampling weight for each dataset during training. For the retrieval database, the entire dataset is used, with the exception of books for which we use a sub-sample of 4%.

### A.2. Wikipedia September 2021

We create an evaluation dataset consisting of 23 Wikipedia articles that were added or heavily edited in September 2021, after we collected our training dataset. In addition, we filter out articles that rely too heavily on templated content, using the method detailed in [§2.6](#) to identify articles with chunks that have a high overlap with their neighbours. [Fig. 10](#) show that little overlap remains between our test dataset and the retrieved neighbours from the training dataset. The full list of included articles is given in [Table 9](#).

Table 9 | Full set of articles included in our **Wikipedia Sept. 2021** evaluation dataset.

Megan Rohrer	Aakashavaani
Emma Raducanu	Junior Eurovision Song Contest 2021
Ambra Sabatini	Pavilion Bukit Jalil
WhyDonate	Blake Desjarlais
The Juggernaut (company)	2021 All-Ireland Senior Football Championship Final
Angela Diaz	Drift-barrier hypothesis
2020 Summer Paralympics	Venomics
2021 Afghan protests	Great Circle (novel)
Rexh Xhakli	Hurricane Ida
Julia Laskin	2021 Montenegrin episcopal enthronement protests
Cuijk	At War With the Silverfish
Ghoubet Wind Power Station	

We first parse articles using `mwparserfromhell`<sup>5</sup>. We then remove sections with the following titles: “references”, “external links”, “sources”, “further reading”, “see also”, “citations”, and “note”. In the remaining sections, we remove Wikilinks and remove the following templates: “reflist”, “notelist”, “notelist-ua”, “notelist-lr”, “notelist-ur”, and “notelist-lg”. We also exclude objects with the “ref” or “table” tag and clean the remaining text with the `strip_code` function. Finally, we concatenate the title and all the sections and use `\n\n` to delimitate them.

## B. Details on the retrieval architecture

We give details on the **RETRO** architecture, and on the fine-tuning procedure we use for **RETROfitting** existing language models.

### B.1. RETRO architecture and implementation

#### B.1.1. Feed-forward architecture

As mentioned in the main text, the overall encoder-decoder architecture is fully feed-forward. We start with a sequence  $X \in \mathbb{V}^n = (C_u)_{1 \leq u \leq l}$ , and its pre-computed neighbours  $(\text{RET}(C_u))_{1 \leq u \leq l}$  and returns logits in  $\mathbb{R}^{n \times |\mathbb{V}|}$ . Along with **ATTN**, **FFW**, **Cca** and **CA** operators introduced in the main text, we define the decoder embedding layer **EMB** :  $\mathbb{V}^n \rightarrow \mathbb{R}^{n \times d}$ , the **SPLIT** operator that extracts chunked intermediary embeddings  $\text{SPLIT}(H) \triangleq (H_u)_{1 \leq u \leq l} \in \mathbb{R}^{l \times m \times d}$  and the read-out layer **READ** :  $\mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times |\mathbb{V}|}$ . We then describe the forward pass in [Algorithm 1](#). In addition to the usual Transformer ones, **RETRO** architecture hyperparameters involves the layer indices  $P_{\text{enc}}$  and  $P$ , at which the encoder and the decoder perform cross-attention.

#### B.1.2. Relative positional encoding in the chunked cross-attention layer

The **CA** operator uses relative positional logits, that are computed from a specific relative distance separating data tokens from retrieval tokens. Indeed, we expect any retrieval neighbour  $\text{RET}(C_u)^j$  and the chunk  $C_u$  to be relatively well aligned, and assume that they start at the same position. Therefore, when computing  $\text{CA}(H_u^+, E_u)$ , we set the distance between the data token  $i \in [1, l]$  of chunk  $C_u^+$  and

<sup>5</sup><https://github.com/earwig/mwparserfromhell>

the retrieval token  $i' \in [1, 2l]$  of  $\text{RET}(C_u)^j$  to be

$$d(i, i') \triangleq i - i' + l - 1. \quad (6)$$

When computing the encoder cross-attentions  $\text{CA}(\text{RET}(C_u)^j, H_u)$ , we set the distance between the retrieval token  $i' \in [1, 2l]$  and the data token  $i \in [1, l]$  to be

$$d_{\text{enc}}(i', i) \triangleq i' - i. \quad (7)$$

Positional logits are obtained as a linear transform of a cosine vector computed from  $(d(i, i'))_{i, i'}$ , and are added to content logits, as in a regular self-attention block.

### B.1.3. Chunked cross-attention implementation

Our implementation of the CCA operator, shown in Listing 1, is based on a vectorized application of a cross-attention layer. For simplicity, we omit the multi-head attention logic and use the simplest Q,K,V attention. We omit relative positional logits computation, described above.

### B.1.4. Optional sharing of embedding matrices

We use disjoint embeddings for the encoder and decoder by default, which allows us to use a different dimensionality for the encoder (typically kept at  $d_{\text{ENC}} = 896$ ) and for the decoder (that we scale up to  $d = 8192$ ). It is possible to share the embeddings, with little difference in training, as we show in the ablation section.

## B.2. Baseline to RETRO model fine-tuning

As shown in Fig. 5, we found that we were able to take a pre-trained baseline transformer and add RETRO through fine-tuning. In all cases, we froze all weights from pre-training and freshly initialised the retrieval encoder and cross-attention weights. In all cases, the cross-attention is added every third layer starting at layer six. The learning rate for the three smaller models was set to  $2 \times 10^{-4}$  and half that for the larger model. We experimented with allowing the entire model to resume training during fine-tuning but consistently found that the best approach was to freeze the pre-trained model. This kept the retrieval-off performance frozen whereas when all weights were tuned the retrieval off performance would degrade.

## C. Training details and hyperparameters

We provide the hyperparameters used in the various experiments of §4.

### C.1. Language model pre-training

In Table 10, we show the hyperparameters of the different models we train. In all cases, we train for 419,430,400,000 training tokens. The three smaller models are trained with a batch size of 256 and the largest model is trained with a batch size of 1024. The minimum learning rate is set to 0.1 times the maximum learning rate, which is shown in Table 10. The learning rate is decayed using a cosine cycle length that matches the total number of training tokens. All models are trained using AdamW (Loshchilov and Hutter, 2019) with a weight decay parameter of 0.1. The learning rate linearly increases from  $10^{-7}$  to the maximum learning rate over the first 750 steps of training. All models use ZeRO to shard the optimiser state (Rajbhandari et al., 2020). Additional infrastructure details can be found in Rae et al. (2021).

**Listing 1** | Jax implementation of the **chunked cross attention**, simplified.

```
n = 128 # Sequence length
m = 16 # Chunk length
r = 32 # Retrieval length
k = 4 # Number of neighbours
d = 16 # Embedding size
l = n // m # Number of chunks

# Parameters
Q = jnp.zeros((d, d))
K = jnp.zeros((d, d))
V = jnp.zeros((d, d))

def relative_positional_encodings(attending_length, attended_length):
    # Classical relative positional encodings
    ...

def cross_attention(chunk, neighbour):
    m, d = chunk.shape
    r, d = neighbour.shape
    queries = chunk @ Q
    keys = neighbour @ K
    logits = queries @ keys.T
    values = neighbour @ V
    return logits, values

def multi_neighbour_cross_attention(chunk, neighbours):
    m, d = chunk.shape
    k, r, d = neighbours.shape

    logits, values = jnp.vectorize(cross_attention,
                                    signature='(m,d), (r,d) -> (m,r), (r,d)')(
        chunk, neighbours)
    assert logits.shape == (k, m, r)
    assert values.shape == (k, r, d)
    logits += relative_positional_encodings(m, r)[None, :, :]
    logits = jnp.moveaxis(logits, 0, -1).reshape((m, r * k))
    values = jnp.moveaxis(values, 0, 1).reshape((r * k, d))
    return jax.nn.softmax(logits) @ values

def multi_chunk_cross_attention(observation, neighbours):
    attending_chunks = jnp.pad(observation[m-1:], [
        ((0, m - 1), (0, 0)),
        mode='constant').reshape(l, m, d)
    chunked_output = jnp.vectorize(multi_neighbour_cross_attention,
                                    signature='(m,d), (k,r,d) -> (m,d)')(
        attending_chunks, neighbours)
    assert chunked_output.shape == (l, m, d)
    output = jnp.pad(chunked_output.reshape(n, d),
                     ((m - 1, 0), (0, 0)),
                     mode='constant')[:n]
    return output

observation = jnp.zeros((n, d)) # Input
neighbours = jnp.zeros((l, k, r, d))

h = multi_chunk_cross_attention(observation, neighbours)

assert h.shape == (n, d) # Output
```

Table 10 | **RETRO model hyperparameters**, along with the size of the decoder.

Baseline	$d_{model}$	$d_{ffw}$	# heads	Head size	# layers	$P$	$P_{ENC}$	Max LR
247M	896	3584	16	64	12	[6, 9, 12]	[1]	$2 \times 10^{-4}$
564M	1536	6144	12	128	12	[6, 9, 12]	[1]	$2 \times 10^{-4}$
1,574M	2048	8192	16	128	24	[9, 12, ..., 24]	[1]	$2 \times 10^{-4}$
7,505M	4096	16384	32	128	32	[9, 12, ..., 32]	[1]	$1 \times 10^{-4}$

Table 11 | Hyperparameters for the Wikitext103 experiments presented in [Table 4](#). We use the same learning rate schedule for the baseline and the RETRO-fitting. For RETRO-fitting, we reset the schedule i.e. the schedule starts from step 0, not from step 35,000.

Model	Number of layers	18
$d$	1024	
$d_{FFW}$	4096	
Key size	64	
Value size	64	
Number of heads	16	
Training data	Dataset	Wikitext103train
	Sequence length	3072
	Batch size	128
	Tokenizer vocabulary size	128,000
Optimisation	optimiser	Adam
	Adam's $\beta_1$	0.9
	Adam's $\beta_2$	0.95
	Adam's $\epsilon$	1e-8
	Dropout rate	0.25
Schedule	Learning rate start	1e-7
	Learning rate max	2.5e-4
	Learning rate min	2e-5
	Warmup steps	4,000
	Cosine cycle steps	100,000
Evaluation	Overlapping proportion	87.5 %

## C.2. Wikitext103 comparison

We provide more details on our Wikitext103 results presented in [§4.1](#) and [Table 4](#). We train a baseline transformer on the Wikitext103 training set with the hyperparameters presented in [Table 11](#). The learning rate ramps linearly from  $1 \times 10^{-7}$  to  $2.5 \times 10^{-4}$  in the first 4,000 steps, then decays to  $2 \times 10^{-5}$  at 100,000 steps using a cosine schedule. The baseline checkpoint at step 35,000 has the lowest perplexity on Wikitext103 valid, of 21.58, for overlapping proportion of 75% (sliding window evaluation that only uses probabilities for tokens that have at least 75% of the sequence length of context, when available). We use this checkpoint for all our baseline and kNN-LM numbers reported in [Table 4](#), except that [Table 4](#) reports for an overlapping proportion of 87.5 %, which slightly lowers the perplexity of our baseline to 21.53 on Wikitext103 valid.

We also use the 35,000 step baseline checkpoint as initialization for a RETROfit, which otherwise uses the same optimiser and schedule hyperparameters but only trains the new retrieval weights, as explained in [§4.2](#). Our best RETROfit checkpoint has a Wikitext103 valid perplexity 18.46, when retrieving from Wikipedia. We use this RETRO checkpoint in [Table 4](#) for all other retrieval sets. The evaluation curves for our baseline and RETROfit is shown if [Fig. 7](#) (left). In this particular case,

because Wikitext103 is quite small, training a RETRO model from scratch led to weaker results than the baseline, at least when retrieving from Wikipedia, as we couldn't find an effective way to mitigate the increased over-fitting due to the additional weights of RETRO.

We also re-implement kNN-LM using the same tokenizer and dataset that we use for our baseline and RETROfitting experiments. kNN-LM has probabilities  $p_{kNN-LM} = \lambda p_{LM} + (1 - \lambda)p_{kNN}$  with  $p_{kNN}(n_k) \propto \exp(-\alpha d_k)$ . To tune  $\lambda$  and  $\alpha$ , we begin with  $\alpha = 0.0012$ , which corresponds to the inverse of the standard deviation of the norm of the embeddings that we use as keys and queries for kNN-LM. We find the best  $\lambda = 0.118$ . We then find the best  $\alpha = 0.00785$  for that value of  $\lambda$ . Fig. 7 center and right respectively show the perplexity of kNN-LM as a function of  $\lambda$  and  $\alpha$ .

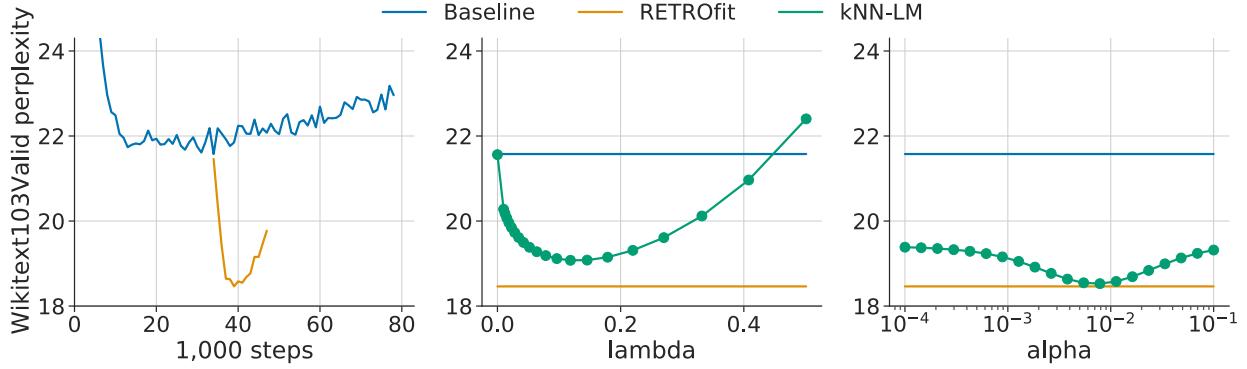


Figure 7 | **Wikitext103valid perplexities.** *Left:* Baseline and RETROfit (initialized from baseline’s checkpoint at 35,000 steps) perplexities as a function of training steps. *Center and right:* kNN-LM perplexity as a function of  $\lambda$  (for  $\alpha = 0.0012$ ) and  $\alpha$  (for  $\lambda = 0.12$ ) respectively.

### C.3. RETROfitting baseline models experiments

In Table 12, we give the hyperparameters used for RETROfitting the models on Massive Text.

Table 12 | Hyperparameters for the RETROfitting experiments

Model	Layers with RETRO-block ( $P$ )	Learning rate	Batch size
172M	Every 3 <sup>rd</sup> from 6	$2 \times 10^{-4} \rightarrow 2 \times 10^{-5}$	256
425M	Every 3 <sup>rd</sup> from 6	$2 \times 10^{-4} \rightarrow 2 \times 10^{-5}$	256
1.5B	Every 3 <sup>rd</sup> from 6	$2 \times 10^{-4} \rightarrow 2 \times 10^{-5}$	256
7.5B	Every 3 <sup>rd</sup> from 6	$1 \times 10^{-4} \rightarrow 1 \times 10^{-5}$	256

### C.4. Question answering experiments

We fine-tune our 7.5B RETRO model for 25,000 steps, using a batch size of 128, a learning rate cosine scheduled from  $10^{-6}$  to  $10^{-7}$ , with a linear ramp of 750 steps. We use dropout in the decoder only, as it performs better than using dropout in both the encoder and the decoder. Each neighbour is formatted as title: {title}, source: {source}. We use the top 20 neighbours from DPR when training and evaluating.

Table 13 | **Performance of RETRO for different variants.** Model performance on C4 evaluation set, measured in bytes-per-bits, for a 247M parameter model trained with a 157 billion token schedule.

Ablation group	Ablation	C4 eval bpb
Model	RETRO	0.822
	No query conditioning	0.829
	No CA positional encodings	0.826
	Shared embeddings	0.823
	6-layer encoder	0.821
Retrieval values	Neighbours N	0.950
	Continuations F	0.895
	No retrieval	0.987
Training neighbours	1 training neighbours	0.858
	4 training neighbours	0.847
Cross attention position	CA top layer (1/12)	0.827
	CA mid layer (6/12)	0.823
	CA top layer (12/12)	0.831
	CA all layers	0.860
	CA every 3 from 1	0.823

## D. Model ablations

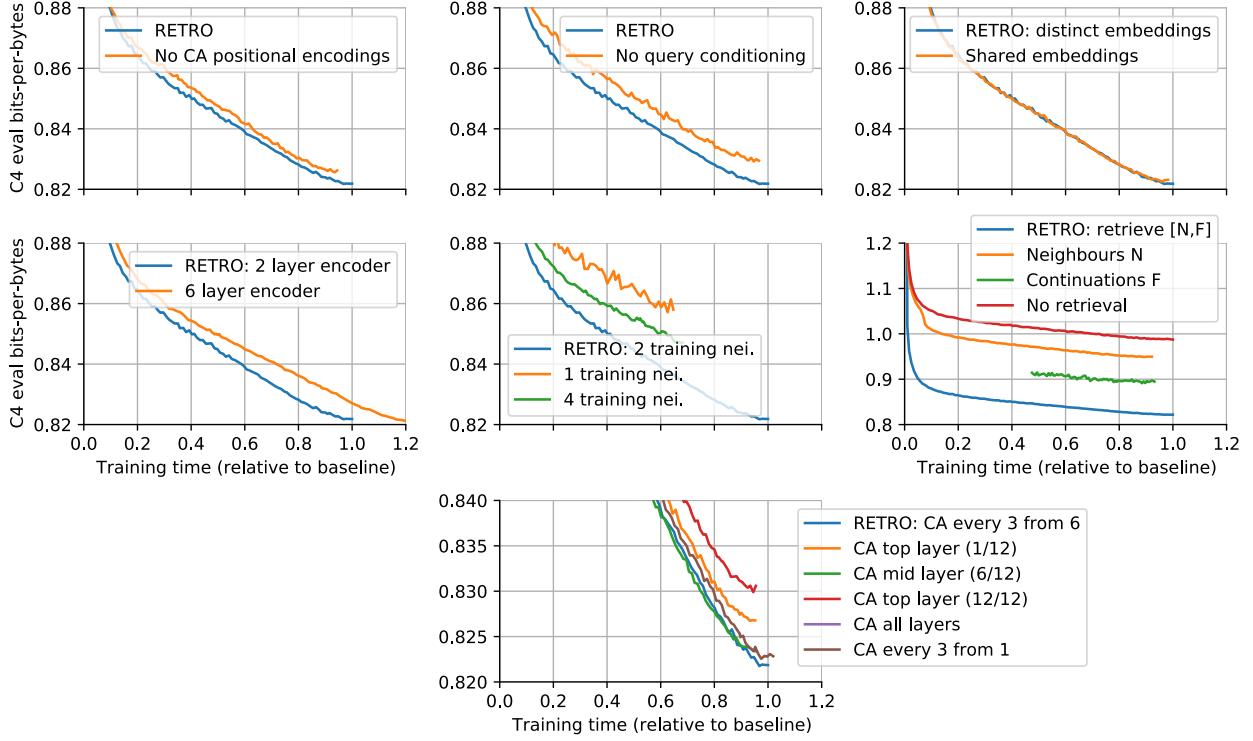
We validate important design choices by evaluating what happens when we do not include them. We use the 247M parameter model for all experiments and we train on a compressed 157 billion token schedule for all ablation experiments. We describe results relative to the default settings presented in the main text and recalled here. We report C4 evaluation loss at the end of the training process, and also compares how the evaluation loss decrease versus the training time, measured relatively to the baseline training time. Results are reported in Fig. 8 and Table 13.

**Using relative encodings in cross-attention.** Using relative encodings in cross-attention, as described in §B.1.2, provides a pure improvement both in the number of steps to reach a given performance and computational efficiency.

**Conditioning the encoder on the previous chunk.** Conditioning the encoder on the previous chunk’s intermediate embeddings, as described in §B.1.1, provides a pure improvement both in term of number of steps and computational efficiency.

**Sharing embeddings.** Sharing embeddings across the encoder and the decoder does not affect performance. This motivates us using separate embeddings, as it allows to have a narrower encoder than decoder as we scale up the decoder size.

**Attending neighbours and their continuation.** RETRO models are trained by attending, for a given chunk, to both the neighbours of the preceding chunk and their continuation in time. We measure how training and evaluating RETRO models on neighbours only and their continuation only affects performance. Overall, attending to neighbours only provides 22% of the performance improvement due to retrieval in RETRO, while attending the future of the neighbours gives 56% of



**Figure 8 | Computational efficiency for different variants.** We report the training curves plotting C4 evaluation bytes per bits against time, relative to the time taken to train the baseline RETRO model. Overall, our design choices are optimal in term of computational efficiency.

the performance. Attending to both neighbours and their continuation is the most efficient choice both in term of final performance and training efficiency.

**Training a deeper encoder.** All models in the text use a relatively small RETRO encoder. We experimented with a 3× deeper encoder. We found that this resulted in a tiny decrease in loss– 0.15% at the cost of a larger training time (+20%). Overall, using a shallow encoder is the best choice in term of training efficiency.

**Training with multiple neighbours.** We measure the effect of training on a single retrieved neighbour, as well as training on 4 neighbours (RETRO uses 2 neighbours in training). Training on a single neighbour results in a large decrease in performance, while training on 4 neighbours does not give substantial performance improvement at the end of training, but induces a large computational overhead. Overall, we find that using 2 neighbours is the best choice in term of training efficiency. Furthermore, evaluation can be done with additional neighbours.

**Frequency of cross-attention.** We measure how the frequency of cross-attention in the decoder affects performance. Overall, attending only once at the top or the bottom layer is a bad choice, while attending once on a mid-depth layer is relatively sound. We choose to have cross-attention every 3 layer as this provides a good trade-off between performance and run-time.

## E. Qualitative experiments

We illustrate the usage of RETRO models by looking at the perplexity of evaluation samples and by producing samples autoregressively.

### E.1. Inspecting neighbours and perplexities on evaluation data

To build an intuition of what kind of information is leveraged by RETRO models, we suggest to have a closer look at a few evaluation documents and the corresponding retrieved data in Tables 16, 17, 18 and 19. In these tables, the 4 rows corresponds to the first 4 chunks of the documents. The left-most column shows the chunk  $C_u$  from the document being evaluated, where each token is coloured by the negative cross entropy loss difference  $L_{\text{RETRO}[\text{OFF}]} - L_{\text{RETRO}}$ , a positive value, coloured in yellow, indicates that RETRO performs better when it has access to neighbours data. The second columns also shows the evaluated chunk  $C_u$  but where each token  $i$  is coloured by the length of the longest common prefix (LCP) with the preceding neighbours, i.e. the largest integer  $j$  such that the prefix  $(x_{i-j-1}, \dots, x_i)$  also appears in  $\text{RET}(C_{u-1})$ . Conversely, columns three and four show the first two neighbours and their continuation, respectively  $[N_u^1, F_u^1]$  and  $[N_u^2, F_u^2]$  coloured by LCP with subsequent chunk  $C_{u+1}$ . LCP colouring helps to visually identify where the evaluated document overlaps the retrieved data. Note that the first chunk,  $C_1$ , in the second column is not coloured as it does not have any preceding neighbours to compute LCP with. Similarly, we do not show the neighbours of the fourth chunk, as these are not used to condition any of the first four chunks.

Our qualitative analysis exhibits two major behaviors.

Firstly, we observe that sometimes, specific facts in  $C_u$  can be extracted from the preceding neighbours  $\text{RET}(C_{u-1})$  and that this can correspond to significant reduction in loss from the RETRO model for the corresponding tokens. Some examples of such behavior include the journal name *Publishers Weekly* in Table 16, the football team name *Tyrone* in Table 17 or the event dates *25 August to 6 September 2020* in Table 18. In these three examples, the evaluated data consists of recent Wikipedia articles written in September 2021, after we built our retrieval dataset (see section §A.2). Yet, relevant information to predict this new data was available in the pre-existing retrieval data and the RETRO model seems to be able to correctly leverage it.

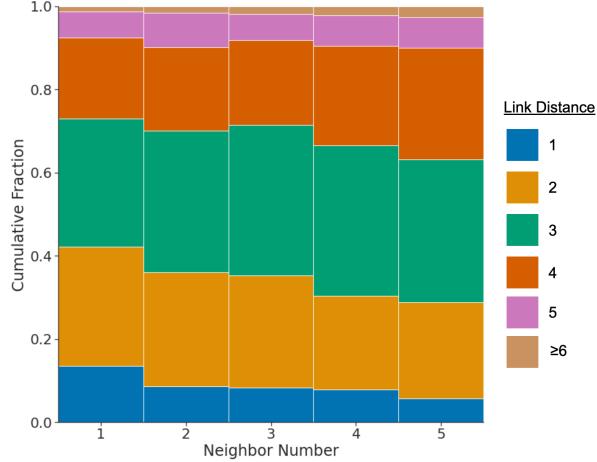
On the other hand, we also observe that some of the evaluation data can partially leak in our training and retrieval data, despite the use of deduplication. RETRO can dramatically exploit such leakage. Table 19 illustrates this behavior, where the chunks  $C_2$  and  $C_3$  largely overlaps  $\text{RET}(C_1)$  and  $\text{RET}(C_2)$  respectively, up to small formatting differences, which leads to much lower RETRO loss for all the corresponding tokens. Fig. 6 shows that it is possible to quantify how much of the RETRO loss reduction is due to each of these two behaviors, by filtering out evaluation chunks that overlaps with the retrieval set.

### E.2. Inspecting samples

We can follow the same procedure as above on samples generated using RETRO models, in order to better understand where retrieval data had an influence on sampling. We show examples of samples obtained using the 7.5B RETRO model in Table 6, 7, 20 and 21.

### E.3. Neighbour quantification

To quantify a notion of distance between the source document and the retrieved chunks, we can ask the distance between source articles when retrieving only from Wikipedia. Consonni et al. (2019)



**Figure 9 | Wikipedia link-distance between retrieved articles.** For each sequences, chunk combination we compute the link distance between the target and the top-5 neighbours using only Wikipedia. The rank shows the relative neighbour distance, where rank-1 is the first neighbour and rank 5 is the fifth. The different colours represent link distance. Because we do not retrieve from the same document, 1 is the smallest value. We find, on average, the distance between random articles with a path between them is over 5.0

provides a Wikipedia link dataset which, for each article, contains a list of neighbouring articles. Using this, we construct a directed graph and compute the distance from one page to another. In Fig. 9 we compute the link-distance between training sequences and the retrieved neighbours. We find that retrieved documents tend to be from articles that are quite close to the article containing the target. Furthermore, we find that on average the distance increases with rank, suggesting that our neighbours are both useful and that the order is reasonable. This provides confidence for our larger-scale experiments where document distance is less well defined.

## F. Complementary quantitative results

We report tables corresponding to quantitative figures of the main text, as well as further filtered language model results on the Pile.

### F.1. Main text datasets

We report the performance of RETRO and baseline models, measured in bits-per-bytes on evaluation set, in Table 14.

### F.2. The Pile

In Fig. 4, we compare RETRO against Jurassic-1 (Lieber et al., 2021). The full bits-per-bytes results are reported in Table 15.

### F.3. Filtered results

**Distribution of leaked chunks in our main evaluation sets.** We evaluate leakage between the evaluation sets and the training set by measuring the proportion of evaluation chunks with a certain

Table 14 | Full results for the main language modelling datasets. First three sets of rows correspond to Fig. 1, last set of rows to Fig. 3.

	Baseline				RETRO [Off]				RETRO [On]			
	172M	425M	1.5B	7.5B	172M	425M	1.5B	7.5B	172M	425M	1.5B	7.5B
C4 Eval bp <sub>b</sub>	0.98	0.92	0.84	0.78	0.98	0.92	0.84	0.78	0.82	0.77	0.71	0.66
C4 Eval bp <sub>b</sub> (900B)	-	-	-	-	-	-	-	-	0.88	0.83	0.76	0.71
C4 Eval bp <sub>b</sub> (360B)	-	-	-	-	-	-	-	-	0.92	0.87	0.80	0.74
C4 Eval bp <sub>b</sub> (180B)	-	-	-	-	-	-	-	-	0.94	0.89	0.81	0.75
C4 Eval bp <sub>b</sub> (90B)	-	-	-	-	-	-	-	-	0.95	0.89	0.82	0.76
C4 Eval bp <sub>b</sub> (36B)	-	-	-	-	-	-	-	-	0.96	0.90	0.83	0.77
C4 Eval bp <sub>b</sub> (18B)	-	-	-	-	-	-	-	-	0.96	0.91	0.83	0.77
C4 Eval bp <sub>b</sub> (9B)	-	-	-	-	-	-	-	-	0.96	0.91	0.83	0.77
C4 Eval bp <sub>b</sub> (4B)	-	-	-	-	-	-	-	-	0.97	0.91	0.84	0.78
C4 Eval bp <sub>b</sub> (2B)	-	-	-	-	-	-	-	-	0.97	0.91	0.84	0.78
C4 Eval bp <sub>b</sub> ( $k = 1$ )	-	-	-	-	-	-	-	-	0.84	0.79	0.73	0.67
C4 Eval bp <sub>b</sub> ( $k = 2$ )	-	-	-	-	-	-	-	-	0.83	0.78	0.72	0.67
C4 Eval bp <sub>b</sub> ( $k = 3$ )	-	-	-	-	-	-	-	-	0.82	0.78	0.71	0.66
C4 Eval bp <sub>b</sub> ( $k = 4$ )	-	-	-	-	-	-	-	-	0.82	0.77	0.71	0.66
C4 Eval bp <sub>b</sub> ( $k = 5$ )	-	-	-	-	-	-	-	-	0.82	0.77	0.71	0.66
C4 Eval bp <sub>b</sub> ( $k = 10$ )	-	-	-	-	-	-	-	-	0.82	0.77	0.71	0.66
C4 Eval bp <sub>b</sub> ( $k = 20$ )	-	-	-	-	-	-	-	-	0.82	0.77	0.71	0.66
C4 Eval bp <sub>b</sub> ( $k = 30$ )	-	-	-	-	-	-	-	-	0.82	0.77	0.71	0.65
C4 Eval bp <sub>b</sub> ( $k = 40$ )	-	-	-	-	-	-	-	-	0.83	0.77	0.71	0.65
C4 Eval bp <sub>b</sub> ( $k = 50$ )	-	-	-	-	-	-	-	-	0.83	0.78	0.71	0.66
C4 Eval bp <sub>b</sub> ( $k = 60$ )	-	-	-	-	-	-	-	-	0.84	0.78	0.72	0.66
C4 Eval bp <sub>b</sub> ( $k = 70$ )	-	-	-	-	-	-	-	-	0.84	0.79	0.72	0.66
C4 Eval bp <sub>b</sub> ( $k = 80$ )	-	-	-	-	-	-	-	-	0.85	0.79	0.73	0.66
C4 Eval bp <sub>b</sub> ( $k = 90$ )	-	-	-	-	-	-	-	-	0.85	0.79	0.73	0.66
C4 Eval bp <sub>b</sub> ( $k = 100$ )	-	-	-	-	-	-	-	-	0.85	0.79	-	0.67
Lambada Accuracy	0.42	0.51	0.61	0.69	0.47	0.54	0.63	0.70	0.52	0.60	0.67	0.73
Curation Corpus bp <sub>b</sub>	0.69	0.63	0.56	0.52	0.68	0.64	0.57	0.51	0.66	0.61	0.55	0.50
Wikitext103 Perplexity	25.62	19.29	13.98	10.65	25.88	19.78	13.89	10.40	3.32	2.96	2.53	2.22
Wikipedia Sept. 2021 bp <sub>b</sub>	0.85	0.78	0.71	0.65	0.86	0.79	0.71	0.65	0.79	0.73	0.66	0.61

overlap  $r(C)$ . We show histograms in Fig. 10. We can see that  $C4$  has some slight overlaps between train and evaluation. Similarly, chunks of Wikitext103 appear in the training set despite having removed the actual Wikitext103 evaluation documents from the training set. On the other hand, our Wikipedia September 21 dataset shows almost no leakage (data being original documents that did not exist at training data creation), and neither does Curation Corpus.

**Filtered results on the Pile.** We report chunk overlap distribution and filtered performance curves on the Pile in Fig. 12 and Fig. 11, respectively. The qualitative interpretation of the filtered curves is the same: RETRO models exploit leakage more, but the performance improvement they provide remains significant even on original chunks that haven't been observed in the training set.

Table 15 | **Full results on The Pile, measured in bits-per-bytes.** Jurassic-1 and GPT-3 numbers are taken from Lieber et al. (2021). Gopher numbers are taken from Rae et al. (2021).

Subset	7B Baseline (Ours)	GPT-3	Jurassic-1	Gopher	7.5B RETRO
arxiv	0.742	0.838	0.680	<b>0.641</b>	0.714
books3	0.792	0.802	0.835	0.706	<b>0.653</b>
dm_mathematics	1.177	1.371	<b>1.037</b>	1.135	1.164
freelaw	0.576	0.612	0.514	0.506	<b>0.499</b>
github	0.420	0.645	0.358	0.367	<b>0.199</b>
gutenberg_pg_19	0.803	1.163	0.890	0.652	<b>0.400</b>
hackernews	0.971	0.975	0.869	0.888	<b>0.860</b>
nih_exporter	0.650	0.612	<b>0.590</b>	0.590	0.635
opensubtitles	0.974	0.932	<b>0.879</b>	0.894	0.930
philpapers	0.760	0.723	0.742	<b>0.682</b>	0.699
pile_cc	0.771	0.698	0.669	0.688	<b>0.626</b>
pubmed_abstracts	0.639	0.625	0.587	0.578	<b>0.542</b>
pubmed_central	0.588	0.690	0.579	0.512	<b>0.419</b>
stackexchange	0.714	0.773	0.655	0.638	<b>0.624</b>
ubuntu_irc	1.200	0.946	<b>0.857</b>	1.081	1.178
uspto_backgrounds	0.603	0.566	<b>0.537</b>	0.545	0.583

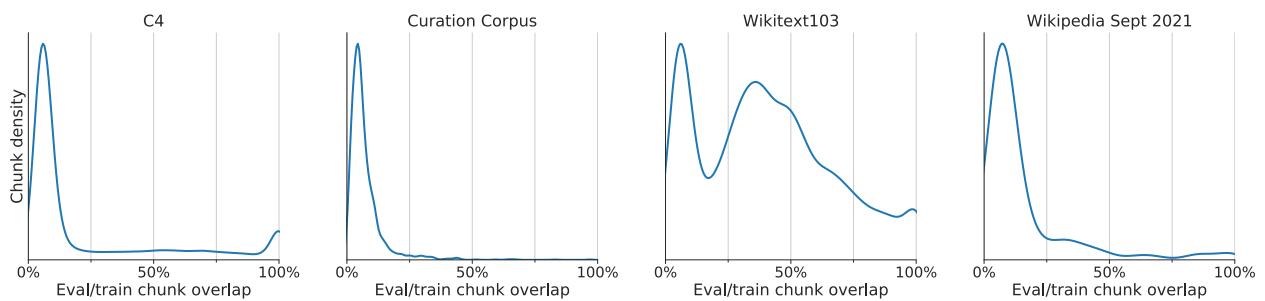


Figure 10 | **Distribution of the overlap between evaluation and train chunks** for C4, Curation Corpus, Wikitext103 and Wikipedia Sept. 2021.

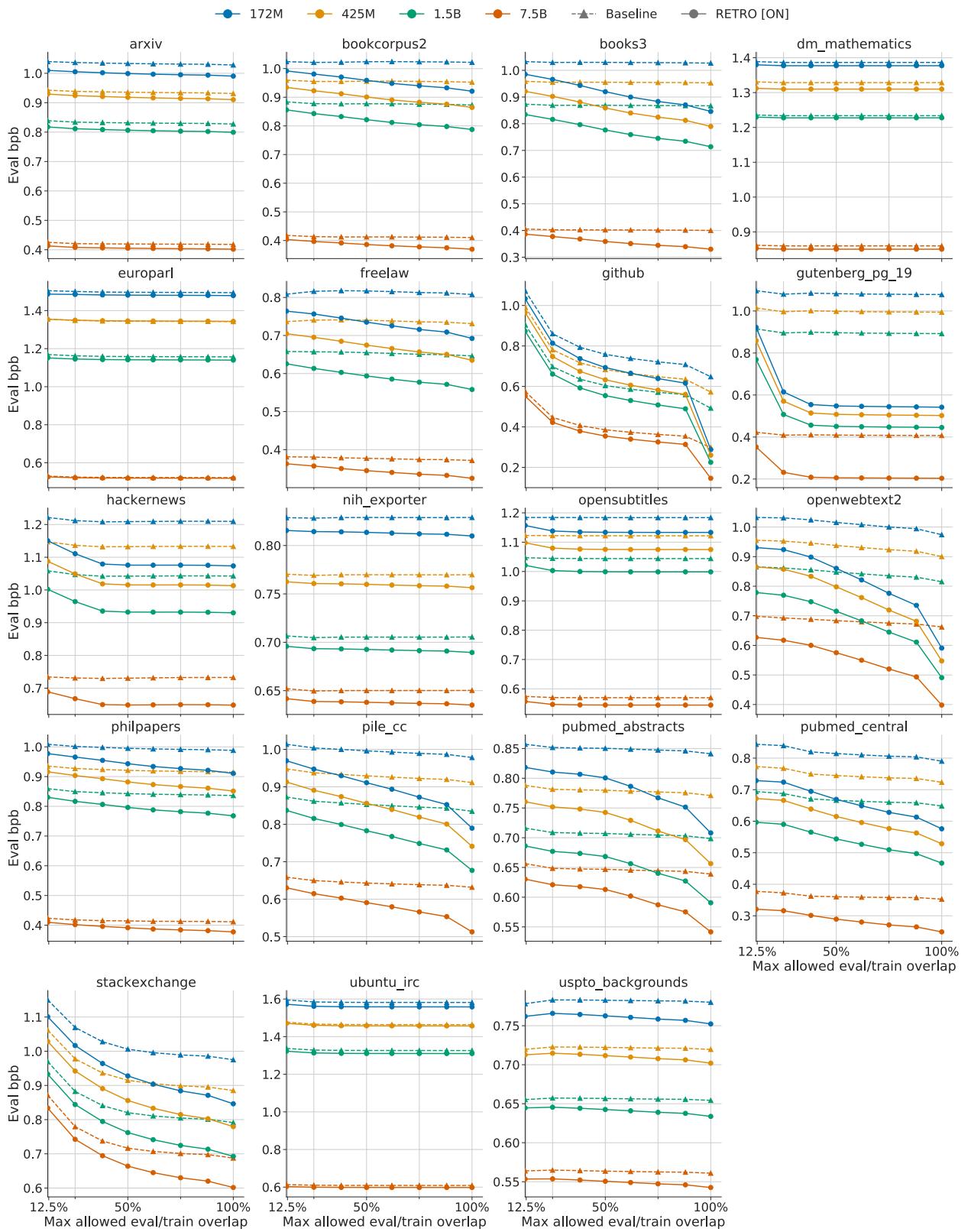


Figure 11 | Filtered evaluation losses on the Pile, with baseline Transformers and RETRO.

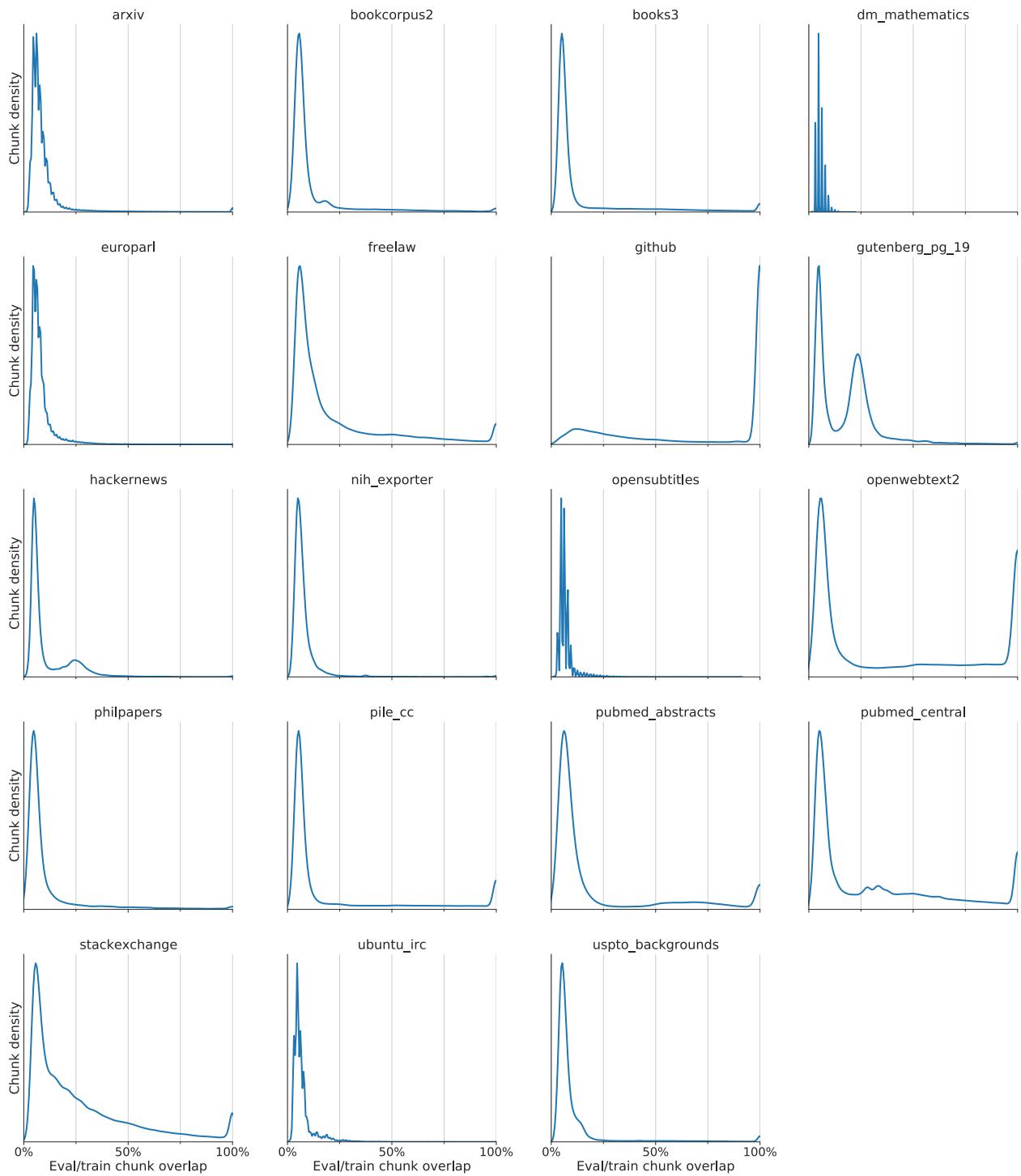


Figure 12 | Distribution of the overlap between evaluation and train chunks for the Pile evaluation sets.

**Table 16 | Great Circle (novel)**, from Wikipedia September 21. The article is about a recent novel and chunks  $C_3$  and  $C_4$  are specifically about its reception. The name **Publishers Weekly** of the journal that reviewed the novel appears both in the neighbours  $[N_3^1, F_3^1]$ ,  $[N_3^2, F_3^2]$  of chunk  $C_3$  and in the subsequent chunk  $C_4$ , where the loss for those tokens is significantly reduced by RETRO.

$C_u$ colored by loss difference $L_{\text{RETRO(OFF)}} - L_{\text{RETRO}} \leq -0.5, = 0, \geq 0.5$	$C_u$ colored by LCP with $\text{RET}(C_{u-1})$ $\text{LCP} = \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \geq 5$	$[N_u^1, F_u^1]$ colored by LCP with $C_{u+1}$ $\text{LCP} = \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \geq 5$	$[N_u^2, F_u^2]$ colored by LCP with $C_{u+1}$ $\text{LCP} = \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \geq 5$
Great Circle (novel) Great Circle is a 2021 novel by Maggie Shipstead, published on May 4, 2021, by Alfred A. Knopf. The novel has been shortlisted for the 2021 Booker Prize. Synopsis The novel consists of two parallel narratives about two fictional women. One is	Great Circle (novel) Great Circle is a 2021 novel by Maggie Shipstead, published on May 4, 2021, by Alfred A. Knopf. The novel has been shortlisted for the 2021 Booker Prize. Synopsis The novel consists of two parallel narratives about two fictional women. One is	The Dutch House (novel) The Dutch House is a 2019 novel by Ann Patchett. It was published by Harper on September 24, 2019. It tells the story of a brother and sister over the course of five decades. The novel was a finalist for the 2020 Pulitzer Prize for Fiction. Plot The Dutch House is a mansion located in Elkins Park, Pennsylvania, a suburb of Philadelphia. It was built in 1922 by the VanHoebeek family, a husband and wife originally from the Netherlands who made their fortune in the tobacco industry. Cyril Conroy, a self-made real estate mogul	The Dutch House (novel) The Dutch House is a 2019 novel by Ann Patchett. It was published by Harper on September 24, 2019. It tells the story of a brother and sister over the course of five decades. [2] The novel was a finalist for the 2020 Pulitzer Prize for Fiction. [3] Plot [edit] The Dutch House is a mansion located in Elkins Park, Pennsylvania, a suburb of Philadelphia. It was built in 1922 by the VanHoebeek family, a husband and wife originally from the Netherlands who made their fortune in the tobacco industry. Cyril Conroy, a self-made real estate mogul
about the disappeared 20th-century aviator Marian Graves, while the other is about the struggling 21st-century Hollywood actress Hadley Baxter, who is attempting to make a film about Marian. Hadley's narrative is told in the first-person, while Marian's sections are told in the third-person	about the disappeared 20th-century aviator Marian Graves, while the other is about the struggling 21st-century Hollywood actress Hadley Baxter, who is attempting to make a film about Marian. Hadley's narrative is told in the first-person, while Marian's sections are told in the third-person	on becoming a filmmaker. She has found a subject for her film project, an obscure African American actress credited only as "the watermelon woman" in old Hollywood films, and the subsequent film recounts her search for this woman even as it covers, in the manner of the earlier Dunyemaries, Dunye's friendships and her love life. In <i>The Watermelon Woman</i> , Dunye makes the film she set out to make in 1990 about African American women artists, a film that both invents an artistic predecessor with whom she can identify and also "finds" Cheryl herself as the artist that she seeks. As Dunye identifies herself	based closely on her own youthful experiences. (She plans the film to be the first of two parts, the second dealing with the aftermath of the first's events.) Byrne plays a young film student named Julie (Hogg's avatar), who starts her artistic education with high hopes of making a movie about a boy named Tony, living in working-class Sunderland, who adores his mother — "is almost obsessed with her," as eager Julie tells her advisers. Her idealism is evident from the start. The advisers are skeptical, and no wonder; Julie's family is posh, with a comfortable country estate and
Reception Great Circle received very favorable reviews, with a cumulative "Rave" rating at the review aggregator website Book Marks, based on 22 book reviews from mainstream literary critics. The novel debuted at number fourteen on The New York Times Hardcover fiction best-seller list for the week ending May	Reception Great Circle received very favorable reviews, with a cumulative "Rave" rating at the review aggregator website Book Marks, based on 22 book reviews from mainstream literary critics. The novel debuted at number fourteen on The New York Times Hardcover fiction best-seller list for the week ending May	first edition hardcover Reception The novel debuted at number one on The New York Times fiction best-seller list. As of the week ending February 20, 2021, the novel has spent 38 weeks on the list. At the review aggregator website Book Marks, which assigns individual ratings to book reviews from mainstream literary critics, the novel received a cumulative "Rave" rating based on 38 reviews, with only one "mixed" review. Publishers Weekly wrote, "Bennett renders her characters and their struggles with great compassion, and explores the complicated state of mind that Stella finds herself in while passing a white." In its	The book also debuted at number two on The New York Times Hardcover Non-fiction best-sellers list on July 28, 2019. [5] It spent eleven weeks on the list. [6] Reception [edit] At the review aggregator website Book Marks, which assigns individual ratings to book reviews from mainstream literary critics, the book received a cumulative "Positive" rating based on 29 reviews: 12 "Rave" reviews, 6 "Positive" reviews, 9 "Mixed" reviews, and 2 "Pan" reviews. [7] Publishers Weekly gave the book a mixed review, writing, "Unfortunately, all three
8, 2021. Critics praised the novel for sustaining its length and for Shipstead's research and intricate novel structure for perfectly interweaving the parallel narratives, despite the time and circumstances separating them. In its starred review, Publishers Weekly wrote, "Shipstead manages to portray both Marian's and Hadley's	8, 2021. Critics praised the novel for sustaining its length and for Shipstead's research and intricate novel structure for perfectly interweaving the parallel narratives, despite the time and circumstances separating them. In its starred review, Publishers Weekly wrote, "Shipstead manages to portray both Marian's and Hadley's		

Table 17 | All-Ireland Senior Football Championship Final, from Wikipedia September 21. The name of the team **Tyrone** appears both in the second neighbours  $[N_u^2, F_u^2]$  of chunk  $C_1$  and in the subsequent chunk  $C_2$ , where the loss for those tokens is significantly reduced by RETRO.

$C_u$ colored by loss difference $L_{\text{RETRO}}[\text{Off}] - L_{\text{RETRO}} \leq -0.5, = 0, \geq 0.5$	$C_u$ colored by LCP with $\text{RET}(C_u - 1)$ $\text{LCP} = \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \geq 5$	$[N_u^1, F_u^1]$ colored by LCP with $C_{u+1}$ $\text{LCP} = \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \geq 5$	$[N_u^2, F_u^2]$ colored by LCP with $C_{u+1}$ $\text{LCP} = \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \geq 5$
2021 All-Ireland Senior Football Championship FinalThe 2021 All-Ireland Senior Football Championship Final was the 134th final of the All-Ireland Senior Football Championship and the culmination of the 2021 All-Ireland Senior Football Championship. The match was played at Croke Park in Dublin on 11 September 2021. It was originally scheduled	2021 All-Ireland Senior Football Championship Final The 2021 All-Ireland Senior Football Championship Final was the 134th final of the All-Ireland Senior Football Championship and the culmination of the 2021 All-Ireland Senior Football Championship. The match was played at Croke Park in Dublin on 11 September 2021. It was originally scheduled	2018 All-Ireland Senior Football Championship FinalThe 2018 All-Ireland Senior Football Championship Final was the 131st final of the All-Ireland Senior Football Championship and the culmination of the 2018 All-Ireland Senior Football Championship in Gaelic football. The match was played at Croke Park in Dublin on 2 September 2018.[3]It was the second time the teams had met in the final; Dublin won the first encounter in 1995. The final was shown live in Ireland on RTÉ Two as part of The Sunday Game live programme, presented by Michael Lyster from Croke Park, with studio analysis from Joe Broally,	2018 All-Ireland Senior Football Championship FinalThe 2018 All-Ireland Senior Football Championship Final was the 131st final of the All-Ireland Senior Football Championship and the culmination of the 2018 All-Ireland Senior Football Championship in Gaelic football. The match was played at Croke Park in Dublin on 2 September 2018. It was the second time the teams had met in the final; Dublin won the first encounter in 1995. It was the third consecutive year that a team qualified under the system of second chances introduced in 2001; Tyrone qualified despite defeat in its provincial championship. Dublin won the final by a margin of six points
for 28 August but had to be postponed by two weeks when the – semi-final was postponed due to a COVID-19 outbreak. Ulster champions Tyrone took on Connacht champions Mayo, in what was their first ever meeting in a final, winning their 4th title after a 2-14 to 0-15 win. Mayo lost	for 28 August but had to be postponed by two weeks when the – semi-final was postponed due to a COVID-19 outbreak. Ulster champions Tyrone took on Connacht champions Mayo, in what was their first ever meeting in a final, winning their 4th title after a 2-14 to 0-15 win. Mayo lost	game 23–23 after extra time, however Ulster progressed under the competition rules as they scored three tries in the match against Leinster's two. The semi-finals took place in mid-November and saw both the away teams win, as Ulster beat Glasgow and Edinburgh beat Connacht. The final was held on Saturday December 2 at Murrayfield Stadium and saw Ulster beat Edinburgh 21–27 to win the Celtic Cup.2004–05 seasonThe format of the competition was changed for the second edition of the competition. The competition was moved to April and May to run after the conclusion of the Celtic League competition, with only eight	with a last-ditch plan of action – play the Munster/Ulster Semi-Final on March 16th, with the winners to play Connacht in the following day's Final. On March 16th then Munster had an easy win over Ulster (9-07 to 0-00) but thankfully for the Munster players, the pitch cut up so badly during the game, it was decided to postpone the following day's hurling Final (until Easter Sunday) with the football Final going ahead on its own on St. Patrick's Day. Less than a week later, on March 23rd, seven
their 11th consecutive final since 1989, losing 6 finals in 9 years, with this latest defeat on an identical scoreline to 2020, when Mayo lost to Dublin. Background were aiming to win their fourth title and first All-Ireland since 1951. Since then, they had lost ten finals (1989, 1996, 1997, 2004, 2006,	their 11th consecutive final since 1989, losing 6 finals in 9 years, with this latest defeat on an identical scoreline to 2020, when Mayo lost to Dublin. Background were aiming to win their fourth title and first All-Ireland since 1951. Since then, they had lost ten finals (1989, 1996, 1997, 2004, 2006,	1-16 to 0-15 winners to qualify for their 10th league final in the past 13 years. They have won seven of their previous league finals under Cuala since 2002, losing the other two to Waterford (2007) and Dublin (2011).Despite the defeat there were some distinct positives from a Galway perspective- most notably the solid displays of Daithí Burke at centre-back, Joseph Cooney at wing-back and Ronan Burke at full-back. Colm Cavanagh continued his excellent form in goal and also hit a stunning free from distance.Indeed it was not the Galway defence that was the problem	which Dublin won by 0-12 to 0-9. Dublin are going for an unprecedented fourth successive Championship win over Kerry. Prior to their current run, which started with the 2011 All-Ireland final, they had only managed two consecutive victories over them on two separate occasions - 1909 and '24, 1976 and '77. The longest winning sequence in the rivalry was set by Kerry between 1941 and 1975, when they won each of the six Championship meetings. Kerry went nine games unbeaten between 1978 and 2009, with four victories either side of a dramatic draw at the quarter-final stage in Thurles in 2001. Sunday will mark their 11th
2012, 2013, 2016, 2017, 2020). appeared in their seventh final, winning on three occasions in 2003, 2005 and 2008. This final was the fifth to be contested by county teams from Connacht and Ulster, the other finals were 1925 (Galway beat Cavan), 1943 (Roscommon beat Cavan), 1948 (Cavan beat	2012, 2013, 2016, 2017, 2020). appeared in their seventh final, winning on three occasions in 2003, 2005 and 2008. This final was the fifth to be contested by county teams from Connacht and Ulster, the other finals were 1925 (Galway beat Cavan), 1943 (Roscommon beat Cavan), 1948 (Cavan beat		

**Table 18 | 2020 Summer Paralympics**, from Wikipedia September 21. The original dates of the event, **25 August to 6 September 2020**, appears both in the neighbors  $[N_1^1, F_1^1]$ ,  $[N_1^2, F_1^2]$  of chunk  $C_1$  and in the subsequent chunk  $C_2$ , where the loss for those tokens is significantly reduced by RETRO. Interestingly, in this case, the neighbors were written at a time when the event hadn't yet been postponed.

$C_u$ colored by loss difference $L_{\text{RETRO}}[\text{OFF}] - L_{\text{RETRO}} \leq -0.5, = 0, \geq 0.5$	$C_u$ colored by LCP with RET ( $C_{u-1}$ ) LCP = <b>0, 1, 2, 3, 4, &gt; 5</b>	$[N_u^1, F_u^1]$ colored by LCP with $C_{u+1}$ LCP = <b>0, 1, 2, 3, 4, &gt; 5</b>	$[N_u^2, F_u^2]$ colored by LCP with $C_{u+1}$ LCP = <b>0, 1, 2, 3, 4, &gt; 5</b>
2020 Summer Paralympics The , brand ed as the Tokyo 2020 Paralympic Game s, was an international multi-sport paraspors event held from 24 August to 5 September 2021 in Tokyo, Japan . They were the 16th Summer Paralym pic Games as organized by the Interna tional Paralympic Committee (IPC).	2020 Summer Paralympics The , brand ed as the Tokyo 2020 Paralympic Game s, was an international multi-sport paraspors event held from 24 August to 5 September 2021 in Tokyo, Japan . They were the 16th Summer Paralym pic Games as organized by the Interna tional Paralympic Committee (IPC).	2020 Summer Paralympics.* The 2020 Summer Paraly mpics are an upcoming major internat ional multi-sport event for athletes with disabilities governed by the Internat ional Paralympic Committee. S cheduled as the 16th Summer Paralym pic Games, it is planned to be held i n Tokyo, Japan from <b>25 August to 6 September 2020</b> . 3. 2019 BWF Para-Bad minton World Championships- The 20 19 BWF Para-Badminton World Champion ships was held from 20 to 25 August 2019 in Basel, Switzerland.- Men's event: Gold Medal: Pramod Bhagat in Singles SL3 Event and Pramod Bhagat and Manoj	2020 Summer ParalympicsThe are an upcoming major international multi-sport event for athletes with disabilities governed by the International Paralympic Committee. Scheduled as the 16th Summer Paralym pic Games, they are scheduled to be held in Tokyo, Japan between 24 August and 5 Sept ember 2021. Originally due to take p lace between <b>25 August and 6 Septem ber 2020</b> . On 24 March 2020, the IOC a nd the Tokyo Organizing Committee officially announced that the 2020 Sum mer Olympics and 2020 Summer Paralympics would be postponed to 2021, due to the COVID-19 pandemic, marking t he first time that the Paralympics h as been postponed. They will still b e publicly marketed as
Originally scheduled to take place f rom 25 August to 6 September 2020, i n March 2020 both the 2020 Summer Ol ympics and Paralympics were postpone d by one year due to the COVID-19 pa ndemic, with the rescheduled Games s till referred to as Tokyo 2020 for m arketing and branding purposes. As with the Olympics, the Games were la rgely held behind	Originally scheduled to take place f rom 25 August to 6 September 2020, i n March 2020 both the 2020 Summer Ol ympics and Paralympics were postpone d by one year due to the COVID-19 pa ndemic, with the rescheduled Games s till referred to as Tokyo 2020 for m arketing and branding purposes. As with the Olympics, the Games were la rgely held behind	once submitted.This process was u ndertaken following the postponement of the Tokyo 2020 Games due to the COVID-19 pandemic, with both the Oly mpics and Paralympics pushed back a year.Now the Tokyo 2020 Olympics are scheduled for July 23 to August 8 while the Paralympics are due to f ollow from August 24 to September 5. The refund process is separate for ticketholders outside of Japan, who purchased tickets through authorise d ticket resellers (ATR).Each ATR has its own individual refund proced ure.Early figures from the refund process for the Tokyo 2020 Olympics stated that around 18 per cent	Olympiad, have now been postponed a nd rescheduled for 23 July to 8 Augu st 2021 in Tokyo, Japan. The Games were postponed in March 2020 as a re sult of the worldwide Covid-19 pande mic, although they will still keep t he name Tokyo 2020 for marketing and branding purposes. This will be th e first time the Olympic Games have been postponed rather than cancelled
closed doors with no outside specta tors due to a state of emergency in the Greater Tokyo Area and other pre fectures. The Games were the second Summer Paralympics hosted by Tokyo s ince 1964, and the third Paralympics held in Japan overall since the 199 8 Winter Paralympics in Nagano. Th e Games featured	closed doors with no outside specta tors due to a state of emergency in the Greater Tokyo Area and other pre fectures. The Games were the second Summer Paralympics hosted by Tokyo s ince 1964, and the third Paralympics held in Japan overall since the 199 8 Winter Paralympics in Nagano. Th e Games featured	has been rescheduled to May 1-4 bec ause of travel restrictions under th e current state of emergency in Toky o and other 10 prefectures across Ja pan.The Tokyo 2020 organizing comm ittee announced that the first of 18 test events for the Olympic and Par allympic Games will involve wheelchai r rugby, which will be held in Yoyog i National Stadium from April 3 to 4 .The FINA Diving World Cup will fo llow from April 18 to 23 at the Toky o Aquatics Centre, which will also s erve as an Olympic qualifying event. The spread of the COVID-19 pandemi c has slowed down in Tokyo three wee ks after the Japanese capital entere d a state of emergency on	Olympic Games, when Tokyo became th e first city in Asia to host the Oly mpic and Paralympic Games, but unfor tunately strong winds made it an imp ossible task this time around.Memb ers of the Tokyo Organising Committ ee of the Olympic and Paralympic Game s (Tokyo 2020), Tokyo Metropolitan G overnment officials, Tokyo 2020 Torc h Relay Official Ambassadors and rep resentatives from Miyagi Prefecture joined the arrival ceremony.FLAME OF RECOVERYThe Olympic flame will now be put on display at various loc ations in the Tohoku region, to high light the message of hope in the are as worst affected by the 2011 Great East Japan Earthqu
539 medal events in 22 sports, with badminton and taekwondo both making their Paralympic debut to replace football 7-a-side and sailing. China topped the medal table for the fifth consecutive Paralympics, with 96 go lds and 207 total medals. Great Brit ain finished second for the ninth t ime,	539 medal events in 22 sports, with badminton and taekwondo both making their Paralympic debut to replace football 7-a-side and sailing. China topped the medal table for the fifth consecutive Paralympics, with 96 go lds and 207 total medals. Great Brit ain finished second for the ninth t ime,		

**Table 19 | Daniel Radcliffe**, from Wikitext103Valid, retrieval data from c4. The chunks  $C_2$  and  $C_3$  are almost entirely retrieved from neighbours  $[N_1, F_1]$  and  $[N_2, F_2]$  respectively, up to formatting differences, which dramatically reduces the loss for these tokens. This example illustrates that when training data leaks into evaluation sets despite deduplication, our RETRO model can directly exploit this leakage.

$C_u$ colored by loss difference $L_{\text{RETRO(OFF)}} - L_{\text{RETRO}} \leq -0.5, = 0, \geq 0.5$	$C_u$ colored by LCP with $\text{RET}(C_u - 1)$ $LCP = \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \geq 5$	$[N_u^1, F_u^1]$ colored by LCP with $C_{u+1}$ $LCP = \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \geq 5$	$[N_u^2, F_u^2]$ colored by LCP with $C_{u+1}$ $LCP = \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \geq 5$
= Daniel Radcliffe =Daniel Jacob Radcliffe ( born 23 July 1989 ) is an English actor who rose to prominence as the title character in the Harry Potter film series. He made his acting debut at 10 years of age in BBC One's 1999 television film David Copperfield, followed by his cinematic debut	= Daniel Radcliffe = Daniel Jacob Radcliffe ( born 23 July 1989 ) is an English actor who rose to prominence as the title character in the Harry Potter film series. He made his acting debut at 10 years of age in BBC One's 1999 television film David Copperfield, followed by his cinematic debut	Daniel Jacob Radcliffe (born 23 July 1989) is an English actor who rose to prominence as the title character in the Harry Potter film series. He made his acting debut at 10 years of age in BBC One's 1999 television film David Copperfield, followed by his cinematic debut in 2001's The Tailor of Panama. At age 11, he was cast as Harry Potter in the first Harry Potter film, and starred in the series for 10 years until the release of the eighth and final film in 2011. Radcliffe began to branch out to stage acting in 2007, starring in the London and New York productions of Equus, and	Daniel Jacob Radcliffe (born 23 July 1989) is an English actor who rose to prominence as the title character in the Harry Potter film series. He made his acting debut at 10 years of age in BBC One's 1999 television film David Copperfield, followed by his cinematic debut in 2001's The Tailor of Panama. At age 11, he was cast as Harry Potter in the first Harry Potter film, and starred in the series for 10 years until the release of the eighth and final film in 2011. Radcliffe began to branch out to stage acting in 2007, starring in the London and New York productions of Equus, and in the
in 2001's The Tailor of Panama. At age 11, he was cast as Harry Potter in the first Harry Potter film, and starred in the series for 10 years until the release of the eighth and final film in 2011. Radcliffe began to branch out to stage acting in 2007, starring in the London and New	in 2001's The Tailor of Panama. At age 11, he was cast as Harry Potter in the first Harry Potter film, and starred in the series for 10 years until the release of the eighth and final film in 2011. Radcliffe began to branch out to stage acting in 2007, starring in the London and New	in 2001's The Tailor of Panama. At age 11, he was cast as Harry Potter in the first Harry Potter film, and starred in the series for 10 years until the release of the eighth and final film in 2011. Radcliffe began to branch out to stage acting in 2007, starring in the London and New York productions of Equus, and in the 2011 Broadway revival of the musical How to Succeed in Business Without Really Trying. He starred in the 2012 horror film The Woman in Black, and played beat poet Allen Ginsberg in the 2013 independent film Kill Your Darlings. He has contributed to many charities,	of Panama. At age 11, he was cast as Harry Potter in the first Harry Potter film, and starred in the series for 10 years until the release of the eighth and final film in 2011. Radcliffe began to branch out to stage acting in 2007, starring in the London and New York productions of Equus, and in the 2011 Broadway revival of the musical How to Succeed in Business Without Really Trying. He starred in the 2012 horror film The Woman in Black, and played beat poet Allen Ginsberg in the 2013 independent film Kill Your Darlings. He has contributed to many charities, including Demelza House Children's Hospice and The Trevor Project. He also made public service announcements for the latter. In 2011, he was awarded the Trevor Project's "Hero Award."
York productions of Equus, and in the 2011 Broadway revival of the musical How to Succeed in Business Without Really Trying. He starred in the 2012 horror film The Woman in Black, and played beat poet Allen Ginsberg in the 2013 independent film Kill Your Darlings. He has contributed to many charities,	York productions of Equus, and in the 2011 Broadway revival of the musical How to Succeed in Business Without Really Trying. He starred in the 2012 horror film The Woman in Black, and played beat poet Allen Ginsberg in the 2013 independent film Kill Your Darlings. He has contributed to many charities,	York productions of Equus, and in the 2011 Broadway revival of the musical How to Succeed in Business Without Really Trying. He starred in the 2012 horror film The Woman in Black, and played beat poet Allen Ginsberg in the 2013 independent film Kill Your Darlings. He has contributed to many charities, including Demelza House Children's Hospice and The Trevor Project. He also made public service announcements for the latter. In 2011, he was awarded the Trevor Project's "Hero Award."	in the 2011 Broadway revival of the musical How to Succeed in Business Without Really Trying. He starred in the 2012 horror film The Woman in Black, and played beat poet Allen Ginsberg in the 2013 independent film Kill Your Darlings. He has contributed to many charities, including Demelza House Children's Hospice and The Trevor Project. He also made public service announcements for the latter. In 2011, he was awarded the Trevor Project's "Hero Award."
including <unk> Hospice Care for Children, and The Trevor Project for suicide prevention among LGBTQ youth, which gave him its Hero Award in 2011. = Early life = Radcliffe was born in West London, England. He is the only child of Alan George Radcliffe, a literary agent, and	including <unk> Hospice Care for Children, and The Trevor Project for suicide prevention among LGBTQ youth, which gave him its Hero Award in 2011. = Early life = Radcliffe was born in West London, England. He is the only child of Alan George Radcliffe, a literary agent, and	including <unk> Hospice Care for Children, and The Trevor Project for suicide prevention among LGBTQ youth, which gave him its Hero Award in 2011. = Early life = Radcliffe was born in West London, England. He is the only child of Alan George Radcliffe, a literary agent, and	including <unk> Hospice Care for Children, and The Trevor Project for suicide prevention among LGBTQ youth, which gave him its Hero Award in 2011. = Early life = Radcliffe was born in West London, England. He is the only child of Alan George Radcliffe, a literary agent, and

**Table 20 | Sample - Déclaration des droits de l'homme: Article premier.** The RETRO[OFF] sample has correct syntax and is almost plausible but is hallucinated. The RETRO[ON] sample is correctly copied from neighbour data, and robustly re-formatted according to our prompt.

Prompt and sample of RETRO[OFF]	Prompt and sample of RETRO[ON] colored by LCP with $C_u-1$ LCP = 0, 1, 2, 3, 4, > 5	[ $N_u^1, F_u^1$ ] colored by LCP with $C_{u+1}$ LCP = 0, 1, 2, 3, 4, > 5	[ $N_u^2, F_u^2$ ] colored by LCP with $C_{u+1}$ LCP = 0, 1, 2, 3, 4, > 5
Article premier - Les hommes naissent et demeurent libres et égaux en droits. Les distinctions sociales ne peuvent être fondées que sur l'utilité commune. Article 2. - Le but de toute association politique est la conservation des droits naturels et	Article premier - Les hommes naissent et demeurent libres et égaux en droits. Les distinctions sociales ne peuvent être fondées que sur l'utilité commune. Article 2. - Le but de toute association politique est la conservation des droits naturels et	de l'homme et du citoyen. Article 1erLes hommes naissent et demeurent libres et égaux en droits. Les distinctions sociales ne peuvent être fondées que sur l'utilité commune. Article 2Le but de toute association politique est la conservation des droits naturels et imprescriptibles de l'homme. Ces droits sont la liberté, la propriété, la sûreté, et la résistance à l'oppression. Article 3Le principe de toute souveraineté réside essentiellement dans la nation. Nul corps, nul individu ne peut exercer d'autorité qui n'en	Les hommes naissent et demeurent libres et égaux en droits. Les distinctions sociales ne peuvent être fondées que sur l'utilité commune. Article 2Le but de toute association politique est la conservation des droits naturels et imprescriptibles de l'Homme. Ces droits sont la liberté, la propriété, la sûreté, et la résistance à l'oppression. Article 3Le principe de toute Souveraineté réside essentiellement dans la Nation. Nul corps, nul individu ne peut exercer d'autorité qui n'en émane expressément. Article 4
imprescriptibles de l'homme, et par conséquent la garantie à chacun des droits suivants : Article 3. - La propriété est un droit inviolable et sacré. Toute personne a le droit de procéder à sa propre conservation. Article 4. - Le	imprescriptibles de l'homme. Ces droits sont la liberté, la propriété, la sûreté et la résistance à l'oppression. Article 3. - Le principe de toute souveraineté réside essentiellement dans la nation. Nul corps, nul individu ne peut exercer d'autorité qui n'en	criptibles de l'homme. Ces droits sont la liberté, la propriété, la sûreté et la résistance à l'oppression. Article 3 - Le principe de toute souveraineté réside essentiellement dans la Nation. Nul corps, nul individu ne peut exercer d'autorité qui n'en émane expressément. Article 4 - La liberté consiste à pouvoir faire tout ce qui ne nuit pas à autrui : ainsi, l'exercice des droits naturels de chaque homme n'a de bornes que celles qui assurent aux autres membres de la société la jouissance de	et imprescriptibles de l'homme. Ces droits sont la liberté, la propriété, la sûreté et la résistance à l'oppression. Article 3 - Le principe de toute souveraineté réside essentiellement dans la Nation. Nul corps, nul individu ne peut exercer d'autorité qui n'en émane expressément. Article 4 - La liberté consiste à pouvoir faire tout ce qui ne nuit pas à autrui : ainsi, l'exercice des droits naturels de chaque homme n'a de bornes que celles qui assurent aux autres membres de la société la jouissance de ces mêmes droits. Ces bornes
but de toute association est la défense des droits de l'homme et du citoyen. Tout citoyen a le droit de participer à l'administration des affaires publiques. Article 5. - L'impunité n'a jamais été et ne sera jamais une fin en elle-même. L'imp	en émane expressément. Article 4. - La liberté consiste à pouvoir faire tout ce qui ne nuit pas à autrui : ainsi, l'exercice des droits naturels de chaque homme n'a de bornes que celles qui assurent aux autres membres de la société la jouissance de ces mêmes	mane expressément. Article 4 - La liberté consiste à pouvoir faire tout ce qui ne nuit pas à autrui : ainsi, l'exercice des droits naturels de chaque homme n'a de bornes que celles qui assurent aux autres membres de la société la jouissance de ces mêmes droits. Ces bornes ne peuvent être déterminées que par la loi. Article 5 - La loi n'a le droit de défendre que les actions nuisibles à la société. Tout ce qui n'est pas défendu par la loi ne peut être empêché, et nul ne peut être contraint à faire ce qu'elle n'	mane expressément. Article 4 - La liberté consiste à pouvoir faire tout ce qui ne nuit pas à autrui : ainsi, l'exercice des droits naturels de chaque homme n'a de bornes que celles qui assurent aux autres membres de la société la jouissance de ces mêmes droits. Ces bornes ne peuvent être déterminées que par la loi. Article 5 - La loi n'a le droit de défendre que les actions nuisibles à la société. Tout ce qui n'est pas défendu par la loi ne peut être empêché, et nul ne peut être contraint à faire ce qu'elle n'
unité, comme le despotisme, s'est toujours révélée être un instrument d'oppression. La tyrannie qui s'est élue juge suprême de la conscience des autres ne peut être jugée. La justice se trouve dans la conscience de chaque citoyen,	droits. Ces bornes ne peuvent être déterminées que par la loi. Article 5 - La loi n'a le droit de défendre que les actions nuisibles à la société. Tout ce qui n'est pas défendu par la loi ne peut être empêché, et nul ne peut être		

**Table 21 | Sample - Decimals of  $\pi$ .** The RETRO[OFF] sample quickly diverges two digits after the end of the prompt whereas RETRO[ON] correctly outputs a large number of  $\pi$  digits, directly copied from the neighbours data.

Prompt and sample of RETRO[OFF]	Prompt and sample of RETRO[ON] colored by LCP with $\text{Ret}(C_u - 1)$ LCP = 0, 1, 2, 3, 4, ≥ 5	[ $N_u^1, F_u^1$ ] colored by LCP with $C_{u+1}$ LCP = 0, 1, 2, 3, 4, ≥ 5	[ $N_u^2, F_u^2$ ] colored by LCP with $C_{u+1}$ LCP = 0, 1, 2, 3, 4, ≥ 5
Pi = 3. 1415926535 8979323846 2643383279 5028841971 69399375 10 5820974944 5923078164 06286 20899 8628034825 3421170679	Pi = 3. 1415926535 8979323846 2643383279 5028841971 69399375 10 5820974944 5923078164 06286 20899 8628034825 3421170679	"1415926535 8979323846 26433 83279 5028841971 693993751058 20974944 5923078164 0628620899 8628034825 34211706798214808 651 3282306647 0938446095 5058 223172 53594081284811174502 8 410270193 8521105559 644622948 9 54930381964428810975 665933 4461 2847564823 3786783 4461 2847564823 3786783	46 2643383279 5028841971 69399 37510 5820974944 592307816406 28620899 8628034825 3421170679 8214808651 3282306647 0938446 095 5058223172 5359408128 4811 174502 8410270193 8521105559 6 446229489 5493038196442881097 56659334461 2847564823 378678 3165 2712019091 4564856692 346 0
8294049602 8988496069 9858349 065 9873246379 9644789435 8628 730709 6540159079 5944069810 5 992965913 7095378412 69378359	8214808651 3282306647 0938446 095 5058223172 53594081284811 174502 8410270193 8521105559 6 446229489 5493038196442881097 5 6659334461 284	651 3282306647 0938446095 5058 223172 5359408128 4811174502 8410270193 8521105559 644622948 89 54930381964428810975 66593 34461 2847564823 3786783165 27 12019091 4564856692 346034861 0 4543266482 1339360726 024914 12737245870066 0631558817 488 1520920 9628292540 91715864	47 0938446095 5058223172 53594 081284811174502 8410270193 85 21105559 6446229489 5493038196 4428810975 6659334461 2847564 823 3786783165 27120190914564 856692 3460348610 4543266482 1 339360726 0249141273724587006 6 0631558817 4881520920 962829 2540 9171536436 7892590360 01 13305305 4882046652 1384146951 94151160943305727036 5759591 953 0921861173 8193261179 3105 118548 0744623799 627495 1227
10 6940372045 7088679512 85612 30857 9046461290 9276642155 56 54603269 5656128798 6366475705 6294954741 5886335339 57657	7564823 3786783165 2712019091 4564856692 3460348610 4543266482 82 1339360726 024914127372458 70066 0631558817 4881520920 962829 28292540 91715	23 3786783165 2712019091 4564 856692 3460348610 4543266482 1 339360726 0249141273724587006 6 0631558817 4881520920 962829 2540 9171536436 7892590360 01 13305305 4882046652 1384146951 94151160943305727036 5759591 953 0921861173 8193261179 3105 118548 0744623799 627495 1227	165 27120190914564856692 3460 348610 4543266482 1339360726 0 2491412737245870066 063155881 7 4881520920 9628292540 917153 64367892590360 0113305305 488 2046652 1384146951 9415116094 3305727036 5759591953 09218611 73 8193261179 310511854807446 23799 6274956735 1885752724 89
76345 5770886953 7988876910 79 66169745 6493974637 6345801550 6663542854 6333764630 6356284 271 7885339804 5672434	364367892590360 0113305305 48 82046652 1384146951 9415116094 3305727036 5759591953 0921861 173 8193261179 31051185480744 623799 6274		

# MEMORIZING TRANSFORMERS

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, Christian Szegedy

{yuhuai, mrabe, delesley, szegedy}@google.com

## ABSTRACT

Language models typically need to be trained or finetuned in order to acquire new knowledge, which involves updating their weights. We instead envision language models that can simply read and memorize new data at inference time, thus acquiring new knowledge immediately. In this work, we extend language models with the ability to memorize the internal representations of past inputs. We demonstrate that an approximate  $k$ NN lookup into a non-differentiable memory of recent (key, value) pairs improves language modeling across various benchmarks and tasks, including generic webtext (C4), math papers (arXiv), books (PG-19), code (Github), as well as formal theorems (Isabelle). We show that the performance steadily improves when we increase the size of memory up to 262K tokens. On benchmarks including code and mathematics, we find that the model is capable of making use of newly defined functions and theorems during test time.

## 1 INTRODUCTION

Transformers (Vaswani et al., 2017) have led to remarkable progress in natural language processing (Devlin et al., 2019; Brown et al., 2020), mathematical reasoning (Polu & Sutskever, 2020; Wang et al., 2020a; Rabe et al., 2021; Li et al., 2021; Hahn et al., 2021; Cobbe et al., 2021), and program synthesis (Austin et al., 2021; Chen et al., 2021; Li et al., 2022). However, transformer performance on many of these tasks is limited by the context length of attention, which is typically short. The ability to attend to far-away tokens is important in many situations. In novels, characters and events are referenced across multiple chapters. In source code, references to classes and functions may occur quite far from the places in which they are defined. In theorem proving, proofs make use of previously defined lemmas.

Attention over long sequences is also useful as a form of rapid learning. Facts and information which are stored in the form of weight matrices must be slowly trained over hundreds of thousands of training steps. By using attention, however, a model can simply *memorize* facts (e.g. function definitions) by storing them as (key, value) pairs in long-term memory, and then retrieve those facts later by creating a query that attends to them. In this case, attention acts as a form of information retrieval, allowing the model to look up facts that it has seen previously.

We demonstrate that a simple and effective way to increase the size of the attention context is to use approximate  $k$ -nearest-neighbor ( $k$ NN) lookup, which is widely used in information retrieval. A number of extremely scalable implementations of  $k$ NN lookup are available, such as ScaNN (Guo et al., 2020) and Faiss (Johnson et al., 2021).

There are two things which distinguish our approach from previous work on long-range attention (c.f. Section 2). First, unlike some other approaches,  $k$ NN lookup does not do averaging or summarization of tokens at long distances, but retrieves exact values even from the distant context.

Second, gradients are not backpropagated into the external memory, which is critical to the scalability of our technique. The keys and values are a function of model parameters, so attempting to backpropagate gradients into external memory would necessarily involve computing all of the keys and values with the current model parameters on every training step. However, if the external memory is not differentiable, then we can instead reuse keys and values that were previously computed on prior training steps, which drastically reduces the amount of computation for large memories. With

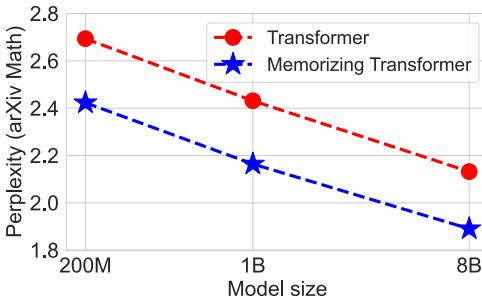


Figure 1: Adding a memory of 8K tokens improves perplexity across different model sizes.

our technique, we are easily able to scale external memory up to sequence lengths of 131k or 262k tokens on a single TPU device, while maintaining a reasonable step time.

We show that model perplexity steadily improves with the size of external memory on a variety of language modelling tasks, including C4 (long documents only), Github code repositories, PG-19 books, formal proofs in Isabelle, and arXiv math papers. We further show that models can generalize to larger memory sizes than they were trained on: models trained with a small memory show gains from using a much larger memory at inference time. Finally, we show that our models are actually using memory in the way that we had hoped, e.g. by looking up the definitions of lemmas in a theorem proving corpus.

The simplicity of the changes to the Transformer architecture allows us to easily integrate this approach into existing code bases, including extremely large language models. We further show that the improvements to quality are maintained across models of increasing size, and that the model improvements gained from adding memory are even larger than increasing the size of the model by 5X or more as shown in Figure 1.

## 2 RELATED WORK

A great deal of work has been done on efficient long-range attention mechanisms; see Tay et al. (2020; 2021) recent surveys. Sliding windows (Beltagy et al., 2020) use a long sequence, but attend within a smaller window, thus reducing complexity to the window size, rather than total sequence length. Approximate mechanisms such as Linformer (Wang et al., 2020b), and Performer (Choromanski et al., 2021) refactor the attention matrix by using a different kernel than softmax to obtain  $O(N)$  complexity. Pooling strategies such as Hierarchical 1D attention (Zhu & Soricut (2021), and Combiner (Ren et al., 2021) apply pooling or averaging over tokens at longer distances. Sparse strategies such as Big Bird (Zaheer et al., 2020) select only a subset of tokens to attend to; Routing Transformers (Roy et al., 2021) use clustering to select the subset, while Reformer (Kitaev et al., 2020) relies on hashing. Hierarchical mechanisms (Ainslie et al., 2020) combine multiple tokens into phrases or sentences to reduce sequence length. Expire-span (Sukhbaatar et al., 2021) prunes far-away tokens that it learns are “unimportant”. (Zemlyanskiy et al., 2021) process long sequences in two passes with different encoders. The second pass is given a lot of context by accessing summaries of the first pass.

Feedback transformers (Fan et al., 2020) use a recurrent architecture in which each token attends to the output of the final layer instead of the previous layer. Recurrence does not increase the size of the attention context itself, but it expands the receptive field at the cost of parallelism and training speed.

Truncated backpropagation through time (Williams & Peng, 1990) was originally introduced as a way of training recurrent neural networks (RNN) over very long sequences, when the entire sequence does not fit in memory. The sequence is chopped into segments, and after each training step, the final RNN state for the segment is saved in a non-differentiable cache, and used as the initial state on the next training step. Neural caches (Grave et al., 2017) extend the cache to contain a record of many prior hidden states, and attend over them. Transformer-XL (Dai et al., 2019) applies this technique to transformers; it caches the (key,value) pairs computed from the previous training step, and uses them as a prefix for the tokens on the next training step, which yields significant gains on long documents. Rae et al. (2020) improve over Transformer-XL by compressing the tokens before adding them to the

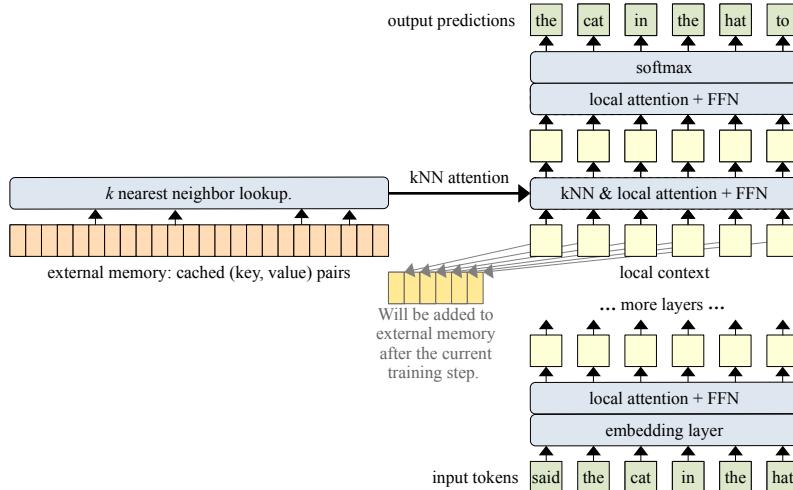


Figure 2: We extend Transformers with access to (key, value) pairs of previously seen subsequences.

cache. In contrast, we use a very large cache without compression, combined with an approximate  $k$ NN attention mechanism over it.

Sukhbaatar et al. (2019) make the observation that the feed-forward portion of a transformer layer functions very much like attention if one replaces the ReLU activation with softmax. They implement a combined attention over both tokens from the input sequence and a learned (and differentiable) “memory”. Lample et al. (2019) exploit this observation to replace the feed-forward layers (FFNs) with a fast  $k$ NN lookup over a much larger “memory”, and achieve large gains in model accuracy without significant computation overhead. (We use  $k$ NN lookup to approximate attention to previous tokens, not to replace the FFN.)

Non-differentiable external memory has been used in different ways by Khandelwal et al. (2020), who run a pre-trained model over an entire corpus, and construct a large table of (key, token) pairs. They then use that table to replace the final softmax layer for token selection in the model, which results in significant improvements in language modeling. Yogatama et al. (2021) extend this approach by a gating mechanism and a process to compress the context into keys for retrieval.

There are several works that combine retrieval with transformers. REALM (Guu et al., 2020), MARGE (Lewis et al., 2020a), RAG (Lewis et al., 2020b), and composite memory for dialog (Fan et al., 2021) retrieve documents from a knowledge base to improve question answering or dialogue. The knowledge base consists of text snippets and is static and typically separate from the inputs and outputs of the models. Instead, we focus on language modeling using a decoder-only model, and propose a simple model that *unifies attention and retrieval*.

$k$ -nearest-neighbor lookup is a general-purpose technique that is used for a wide variety of machine learning and retrieval tasks, and high-performance implementations are available for various architectures (Johnson et al., 2021) (Guo et al., 2020). Memory-efficient Transformers (Gupta et al., 2021) replace dense attention with a  $k$ NN lookup to increase speed and reduce memory usage.

### 3 METHOD

The architecture of our  $k$ NN-augmented transformer is shown in Figure 2. The bulk of the model is a vanilla, decoder-only transformer (Vaswani et al., 2017). The input text is tokenized, and the tokens are embedded into vector space. The embedding vectors are passed through a series of transformer layers, each of which does dense self-attention, followed by a feed-forward network (FFN). Since this is a decoder-only language model, we use a causal attention mask and the token embeddings of the last layer are used to predict the next token.

Long documents are split into subsequences of 512 tokens, and each subsequence is used as the input for one training step. In contrast to standard practice, we do not shuffle the subsequences; instead, each long document is fed into the transformer sequentially, from beginning to end, as is done with Transformer-XL (Dai et al., 2019).

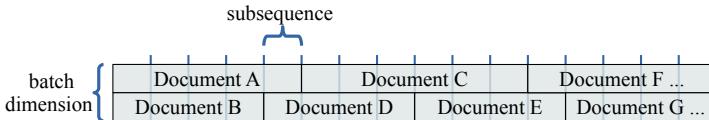


Figure 3: Our data pipeline splits documents into subsequences and packs subsequences into batches.

We also use a Transformer-XL style cache, which holds the keys and values from the previous training step. When doing self-attention, the cached keys and values are prepended to the current keys and values, and we use a sliding-window causal mask (Beltagy et al., 2020) so that each token has a local context that includes the previous 512 tokens.

### 3.1 $k$ NN-AUGMENTED ATTENTION LAYER

One of the transformer layers near the top of the stack is a  *$k$ NN-augmented attention layer*, which combines two forms of attention. Like all of the other layers, it uses standard dense self-attention on the *local context*, which is the input subsequence for the current training step. Unlike the other layers, however, it also does an approximate  $k$ -nearest-neighbor search into the *external memory*.

The same queries are used for both the local context, and for the external memory. The keys and values also belong to the same distribution; after each training step, the (key, value) pairs in the local context are appended to the end of the external memory. If the document is very long, old (key, value) pairs will be dropped from the memory to make room for new ones. Thus, for each head, the external memory keeps a cache of the prior  $M$  (key, value) pairs, where  $M$  is the memory size.

The  $k$ NN lookup will return a set of *retrieved memories*, which consist of the top- $k$  (key, value) pairs that  $k$ NN search returns for each query (i.e. each token) in the input subsequence. As with standard dense attention, we first construct an attention matrix by computing the dot product of each query against the retrieved keys, then apply softmax, and finally return a weighted sum of the retrieved values. Unlike standard dense attention, the retrieved memories contain a different set of (key, value) pairs for each query.

Attention over the local context is performed in the usual way. The results of  $k$ NN-attention and local attention are then combined using a learned gate:

$$g = \sigma(b_g) \quad (1)$$

$$\mathbf{V}_a = \mathbf{V}_m \odot g + \mathbf{V}_c \odot (1 - g) \quad (2)$$

where  $\sigma$  is the sigmoid function, and  $\odot$  is element-wise multiplication.  $\mathbf{V}_a$  is the combined result of attention,  $\mathbf{V}_m$  is the result of attending to external memory, and  $\mathbf{V}_c$  is the result of attending to the local context. The *bias*  $b_g$  is a learned per-head scalar parameter, which allows each head to choose between local and long-range attention. In our experiments, the value of the gate  $g$  does not depend on the content of the token at each position, although that would be a trivial extension to implement. We did observe that over time, most heads learned to attend almost exclusively to external memory.

**Position bias.** For dense attention within the local context, we use the T5 relative position bias (Raffel et al., 2020). As noted by Dai et al. (2019), adding a global position encoding to each token does not work well when processing long documents. We don't use a position bias for the retrieved memories. Experiments on the PG19 dataset (Sun et al., 2021) have shown that relative position does not appear to matter at long range, and the T5 relative bias puts all long-range tokens in the same bucket anyway.

**Batching.** Figure 3 illustrates how multiple long documents of different lengths are packed into a batch, and split into subsequences. Each subsequence in the batch comes from a different document, and thus requires a separate external memory, which is cleared at the start of each new document.

### 3.2 DISTRIBUTIONAL SHIFT

Because each long document is processed over multiple training steps, there is a distributional shift in the keys and values that are stored in external memory. The model parameters that produce the queries change over time, and will thus have shifted since the keys and values were stored. For very large memories, older records may become "stale." Similar observations have been made for CrossBatch memory (Wang et al., 2020c) in the vision domain.

To reduce the effects of staleness, we normalize keys and queries (Henry et al., 2020). Normalization does not eliminate staleness, but it at least ensures that older keys and newer keys do not differ in magnitude. We also found that normalization helps stabilize training with the Transformer-XL cache.

In some of our experiments, we observed that training models from scratch with a large memory sometimes resulted in worse performance than pretraining the model with a small memory of size 8192, and then finetuning it on a larger memory. This training instability could be due to staleness. However, models seem to be able to cope with a limited degree of staleness (with the small memory) by adjusting their queries accordingly.

### 3.3 APPROXIMATE $k$ NN

We employ *approximate*  $k$ NN search rather than exact  $k$ NN search because it significantly improves the computational speed of our model. We use a simple approximation of  $k$ NN for TPUs, which has a recall of about 90%, i.e. 90% of the true top  $k$  are returned in the approximate top  $k$ . There are various other efficient approximate  $k$ NN algorithms available for CPU and GPU/TPU, for example through Faiss (Johnson et al., 2021) or ScaNN (Guo et al., 2020), which can scale into the billions.

## 4 EXPERIMENTS

We evaluate the effect of adding external memory on five language modeling tasks, all of which involve long-form text: English language books (PG-19), long web articles (C4), technical math papers (arXiv Math), source code (Github), and formal theorems (Isabelle). The results show significant improvements in the perplexity of the model with the addition of external memory. We experimented with various sizes of external memory, from 1536 to as high as 262K. On most of the datasets, there was an initial sharp gain from adding a small external memory, followed by smaller but steadily increasing gains as the size of the memory was increased.

### 4.1 DATASETS

**arXiv Math** For the arXiv dataset, we collected a corpus of papers by downloading them via the arXiv Bulk Data Access<sup>1</sup>. We filtered papers to include only articles labeled as “Mathematics” and whose L<sup>A</sup>T<sub>E</sub>X source was available. The number of tokens per paper in this dataset is roughly comparable to the number of tokens per book in PG19, because L<sup>A</sup>T<sub>E</sub>X source has many special characters and the tokenizer tends to output small subwords.

**Github** We used BigQuery<sup>2</sup> to obtain a large corpus of Github repositories that are published with open-source licenses. We used file endings to filter for files in the languages C, C++, Java, Python (including Jupyter notebooks), Go, and TypeScript. Individual source code files are often fairly short, and there are many dependencies and cross-references between files in the repository. To capture these dependencies, we created one long document for each Github repository by traversing the directory tree, and concatenating all of the files within it. The order in which files are traversed within the repository is random, but each subdirectory is processed as a unit, so that all the files within the subdirectory are close to each other in the resulting document. Source code is usually structured so that related files are all grouped together in the same subdirectory; this traversal preserves that structure, while still shuffling files and subdirectories in random order.

**Formal Math – Isabelle** The Isabelle corpus consists of formal mathematical proofs of theories. We collected all 627 theories available on The Archive of Formal Proofs<sup>3</sup> (as of October 6, 2021) and an additional 57 theories from the Isabelle standard library<sup>4</sup> to create a corpus of 684 theories. All theories have open-source licenses. Each theory is a self-contained mathematical object, on topics such as foundational logic, advanced analysis, algebra, or cryptography, and consists of multiple files containing proofs. As with the Github corpus, all files that make up a theory are concatenated

<sup>1</sup>[https://arxiv.com/help/bulk\\_data](https://arxiv.com/help/bulk_data)

<sup>2</sup><https://console.cloud.google.com/marketplace/product/github/github-repos>

<sup>3</sup><https://www.isa-afp.org/topics.html>

<sup>4</sup><https://isabelle.in.tum.de/>

Context	Memory	XL cache	arXiv	PG19	C4(4K+)	GitHub	Isabelle
512	None	None	3.29	13.71	17.20	3.05	3.09
2048	None	None	2.69	12.37	14.81	2.22	2.39
512	None	512	2.67	12.34	15.38	2.26	2.46
2048	None	2048	2.42	11.88	14.03	2.10	2.16
512	1536	None	2.61	12.50	14.97	2.20	2.33
512	8192	None	2.49	12.29	14.42	2.09	2.19
512	8192	512	2.37	11.93	14.04	2.03	2.08
512	65K	512	2.31	11.62	14.04	1.87	2.06
2048	8192	2048	2.33	11.84	13.80	1.98	2.06
2048	65K	2048	<b>2.26</b>	<b>11.37</b>	<b>13.64</b>	<b>1.80</b>	<b>1.99</b>

Table 4: Average token-level perplexities of each model when trained for 500k steps.

together into one long document. Unlike the Github corpus, we order the files according to their import dependencies, so that later files use sub-theorems that are proved in earlier files.

**C4(4K+)** C4, the colossal cleaned common crawl, is a very large collection of documents that have been scraped from the internet (Raffel et al., 2020). We filtered out all documents that have less than 4096 tokens to focus on documents where memory can have an impact.

**PG-19** PG-19 is a large dataset of English-language books, published prior to 1919, which were retrieved from the Project Gutenberg archive (Rae et al., 2020; Sun et al., 2021). PG-19 is one of the few public datasets that only contains full-length books, and has become a benchmark for long-range natural language text modeling.

## 4.2 EXPERIMENTAL METHOD

We used a 12-layer decoder-only transformer (with and without Transformer-XL cache) with an embedding size of 1024, 8 attention heads of dimension 128, and an FFN hidden layer of size 4096. For all of our experiments, we used  $k = 32$ . Unless specified otherwise, we use the 9th layer as the  $k$ NN augmented attention layer. We used a sentence-piece (Kudo & Richardson, 2018) tokenizer with a vocabulary size of 32K.

We used the Adafactor optimizer (Shazeer & Stern, 2018). In preliminary experiments, we conducted a hyperparameter search to determine the optimal learning rate among three choices ( $\{3.0, 1.0, 3 \cdot 10^{-1}\}$ ), and found that 1.0 works best. We used a linear warmup schedule for the first 1000 steps, followed by square root decay. We trained the models from scratch for 500K steps on all the datasets, except for the Isabelle dataset. Isabelle is small, so we stopped training after 100K steps when the model began to overfit. We ran all of our experiments on 32 TPU cores. Our models were implemented in JAX (Bradbury et al., 2018) and Flax (Heek et al., 2020).

When comparing models with different context lengths, we adjusted the batch size (the number of documents in a batch) so that there are always  $2^{17}$  tokens in a batch. E.g., a model with a context length of 512 has a batch size of 256, while the 2048 model has a batch size of 64.

We experimented with multiple implementations of approximate  $k$ NN lookup with different tradeoffs between quality and computational cost. We did not observe a significant degradation of the model quality when switching to lower quality approximations of  $k$ NN, so the model appears to be quite robust with respect to the quality of  $k$ NN retrieval. For a model with around 200M trainable parameters the step time increased from 0.2s to 0.25s when we added a memory of size 8K, and to 0.6s when we added a memory of size 65K (measured on TPUs).

## 4.3 EFFECT OF EXTERNAL MEMORY

**Adding external memory results in substantial gains across datasets and architectures,** as shown in Table 4. Across all five datasets, adding external memory to either the vanilla Transformer or the Transformer-XL architecture improves perplexity by a substantial amount. For example, on

Context	Pretrain	Fine-tune	Perplexity
512	8192	None	2.37
512	65K	None	2.31
512	8192	65K	2.32
512	8192	131K	2.30
512	8192	262K	2.26
2048	8192	None	2.33
2048	65K	None	2.26
2048	65K	131K	2.23
2048	65K	262K	<b>2.21</b>

Table 5: Finetuning for 20K steps to make use of a larger memory on the arXiv data set.

C4(4K+) dataset, adding memory of size 8192 improves the perplexity of the vanilla Transformer (with context size 512) from 17.20 to 14.42, and improves Transformer-XL from 15.38 to 14.04.

**Increasing the size of the memory increases the benefit of the memory.** The best perplexities for all datasets and architectures were obtained with a memory size of 65K.

Note that Transformer-XL with context size 2048 already has a theoretical receptive field that is quite large. Each token in a higher layer can attend up to 2048 tokens away in the layer below, so the total receptive field is  $2048 \cdot 12$  (layers)  $\sim 25K$ . Nevertheless, we still saw a substantial gain when adding an external memory of size 8192 to this model.  $k$ NN attention into memory would appear to be a more effective way to retrieve information from the distant past than the Transformer-XL cache.

On the other hand, we also saw improvements by adding XL cache to the large-memory (65K) models. In a vanilla (non-XL) Transformer, the first few tokens in a sequence have very little context, and thus have higher perplexity. The XL cache provides additional local short-range context at the start of a sequence, which complements the long-range context provided by external memory.

Interestingly, in a vanilla Transformer, using even a small external memory of size 1536 provides a gain in perplexity which is almost as good as using a local context of size 2048 but no memory (e.g. Table 4). This is surprising, because the external memory is not differentiable, and is added only to one layer of the Transformer, whereas increasing the context size is differentiable and affects all layers. We conclude that the lower layers of a Transformer don't necessarily need long-range context, and having a differentiable memory is not as important as one might suspect.

#### 4.4 SCALING TO LARGER MODELS

We scaled up the Transformer model to sizes of 1 and 8 billion parameters. For the 1 billion parameter model, we use 8 layers, 32 heads with head dimension 128,  $d_{\text{model}}$  2048, and  $d_{\text{ff}}$  16384. For the 8 billion parameter model, we use 64 heads, 16 layers,  $d_{\text{model}}$  4096, and  $d_{\text{ff}}$  32768. We used a context size of 2048, memory size of 8192, and no XL cache. We ran the comparisons to the vanilla Transformer on the arXiv math dataset. Scaling plots are shown in Figure 1.

External memory provides a consistent improvement to the model as it is scaled up. Remarkably, we found that **the smaller Memorizing Transformer with just 8k tokens in memory can match the perplexity of a larger vanilla Transformer which has 5X more trainable parameters.**

#### 4.5 FINETUNING ON LARGER MEMORIES

**Finetuning on a larger memory.** In some cases, training was unstable when using large memories, possibly due to distributional shift early in the training (See Section 3.2). Thus, for memories of 131K or more tokens, we first pretrain the model with a memory size of 8192 or 65K for 500K steps, and then finetune it with the larger memory for an additional 20K steps. The results of finetuning on the arXiv Math data set are shown in Table 5. **Increasing the size of external memory provided consistent gains up to a size of 262K.** Note that 262K tokens is longer than almost all of the documents in arXiv, and thus we would not expect to see any gain past this point (see Appendix A).

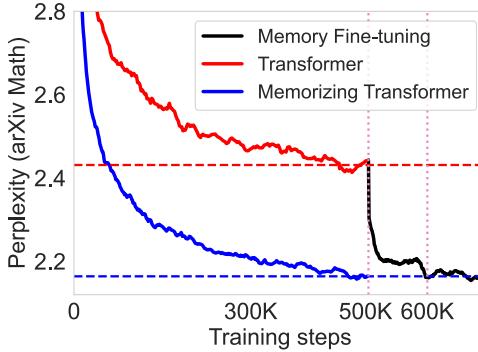


Figure 6: Finetuning a 1B vanilla Transformer model to use external memory of size 65K.

**Finetuning a non-memory model to use memory** Pretraining can be very costly both in time and computational resources. Thus, a natural question to ask is: *can one fine-tune a pretrained Transformer to use external memory?* The answer is yes!

We took a pre-trained 1B vanilla Transformer model, and fine-tuned it to use external memory (the 1B models used in Section 4.4). The fine-tuning result is shown in Figure 6. Notice that the model quickly learns to use external memory. Within 20K steps (4% of the pre-training time) the fine-tuned model has already closed 85% of the gap between it and the 1B Memorizing Transformer, and after 100k steps it has closed the gap entirely.

#### 4.6 INFORMATION RETRIEVAL PATTERNS

We conducted a qualitative study of what the model was actually retrieving from external memory, by finding which tokens showed the biggest improvements in cross-entropy loss when the size of the memory was increased, and then examining the top- $k$  retrieved memories for those tokens. We found that the model gained the most when looking up rare words, such as proper names, references, citations, and function names, where the first use of a name is too far away from subsequent uses to fit in the local context. This result is in keeping with the prior analysis of long-context Transformers on PG19 (Sun et al., 2021), which found similar lookup patterns. For this experiment, we used a slightly older version of the architecture without the gating mechanism.

**Which tokens show a benefit from memory?** Figure 7 shows a visualization of which tokens show an improvement when the size of the external memory is increased. We selected a math paper at random, and plotted the difference in cross entropy loss for each token  $x_i$  in the paper, comparing two models with the same parameters, but with memories of different sizes.  $\Delta_i = \text{cross-entropy}_{8192}(x_i) - \text{cross-entropy}_{32K}(x_i)$ . Positive values show an improvement in loss.

The  $x$ -axis on the chart is the token number  $i$ , while the  $y$ -axis is  $\Delta_i$ . For the first 8192 tokens, the difference between the two models is zero, since the larger capacity of the 32K memory isn't being used yet. However, after token 8193, we can see that the larger memory helps, on average, over the smaller memory. The benefit is not universal, since the predictions for some tokens become worse, possibly due to the fact that a relevant retrieved memory no longer makes it into the top- $k$  when the size of the external memory is increased. This figure also shows that the benefit of external memory is somewhat sparse. The improvement in perplexity seems to be mainly driven by a small percentage of tokens that obtain a large improvement in cross-entropy loss when using the larger memory.

**What information is being looked up?** Given that only a subset of tokens shows improvement from external memory, we did a further investigation into what, exactly, those tokens are using the memory for. We took those tokens which showed the largest improvement in cross-entropy loss, and for each of them tokens, we examined the top- $k$  retrieved memories. We studied arXiv math, Github and Isabelle corpus. For arXiv math and Github, we found the model retrieved function and variable names. See more details with examples in Appendix B.

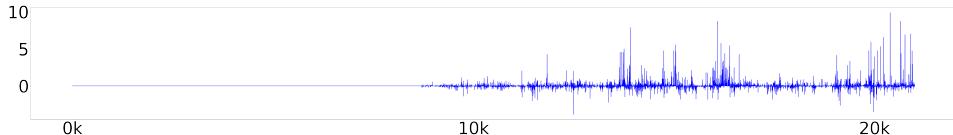


Figure 7: Difference in loss for each token in a randomly chosen paper, using the same model once with a memory size of 8K and once with 32K. Higher numbers mean the longer memory helped in comparison to the shorter memory. This paper is 22K tokens long.

Query index	Input	Target	Surrounding context	Retrieved index	Retrieved surrounding context
29721	mark	ov	rule prob_space. markov_inequality	8088	M. t \langle le> X a} \langle le> expectation X / t"
40919	_	th	= ( subgraph_threshold H n / p n)	27219	threshold H n = n powr (-1 / max_density'
49699	S	w	assumes " orthonormal_system S w"	28050	definition orthonormal_system :: "

Table 8: Examples of memory retrieval in the Isabelle dataset. The model is able to find the definition of a lemma from a reference to it. The retrieved surrounding context (highlighted) is the definition body of the mathematical object highlighted in the querying context.

*Retrieving mathematical definitions.* Our case study on the Isabelle corpus provides one of the clearest illustrations of how a model can make good use of external memory. When predicting the name of a mathematical object or a lemma, the model looked up the definition from earlier in the proof. Examples of this behavior are shown in Table 8. In example 1, the model retrieves a definition within the body of a lemma, `markov_inequality`. In example 2, it retrieves the definition of a previously defined concept `subgraph_threshold`. In example 3, it retrieves the definition of `orthonormal_system`. We manually checked 10 examples where the model made a prediction of lemma names, and 8 out of 10 times model found the body of the lemma it needs to predict. In the other two cases, the model also looked up materials in the immediate vicinity. To the best of our knowledge, this is the first demonstration that attention is capable of looking up definitions and function bodies from a large corpus. The Isabelle case study used a model with two memory layers of size 32K.

## 5 CONCLUSION

We present a simple extension to the Transformer architecture, called  $k$ NN-augmented attention, which dramatically increases the length of the context that a language model can attend to by using  $k$ -nearest-neighbor lookup into a large external memory. We demonstrate the effectiveness of external memory in a series of language modeling experiments over a variety of long-document datasets, including LaTeX documents, source code, formal proofs, and books.

The Memorizing Transformer shows large improvements in perplexity over the baseline for all of the data sets and architectures that we studied; it is comparable to a vanilla transformer that has 5 times the number of parameters. Perplexity continues to improve with increasing memory size, although there is a point of diminishing returns. Moreover, external memory continues to provide benefits even as the transformer is scaled up from 200M to 8B parameters. Perhaps most intriguingly, a Memorizing Transformer does not need to be pre-trained from scratch; it is possible obtain large gains from adding memory to an existing pre-trained model, and then fine-tuning it.

Unlike other forms of attention,  $k$ NN retrieval can be easily scaled up to huge memory sizes, and is thus potentially able to leverage vast knowledge bases or code repositories. How to make the best use of this capability is a topic for future work.

## ACKNOWLEDGMENTS

We want to thank Charles Staats for the many fruitful discussions and detailed comments, Henryk Michalewski for early version of the memory implementation, Petros Maniatis for his help with our code datasets, Aitor Lewkowycz for his help with larger scale memorizing transformer experiments, Behnam Neyshabur for his comments on finetuning non-memory models, Imanol Schlag for his proofread and detailed comments, and Dennis Lee and Manzil Zaheer for discussions about large-scale attention and retrieval.

## A LENGTH OF INPUTS



Figure 9: Histogram of the number of tokens in arXiv math papers dataset. We truncated the histogram at 500k tokens. The maximum paper had almost 1.6M tokens.

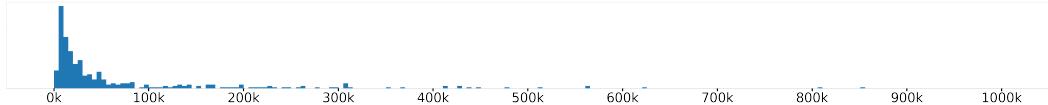


Figure 10: Histogram of the number of tokens in Github repositories dataset. We cut off the long tail of this plot. The repository with the maximum length has just over 9M tokens.

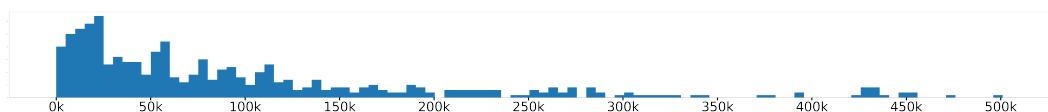


Figure 11: Histogram of the number of tokens in Isabelle proof scripts dataset.

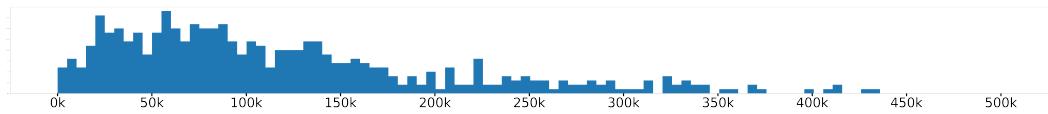


Figure 12: Histogram of the number of tokens in PG19 books dataset.



Figure 13: Histogram of the number of tokens in C4 documents filtered by documents that have less than 4096 tokens.

### A.1 ABLATION STUDIES

In the following section, we performed ablation studies to investigate the effects of various hyperparameters. Unless otherwise specified, we carried out these experiments with a memorizing transformer with context size 512, XL cache 512 with a memory size of 8192.

**Multiple kNN layers.** We experimented with using two  $k$ NN layers, rather than just one. However, we did not see further benefits brought by more than multiple retrieval layers.

**$k$ NN layer index** We experimented with adding the external memory to layer 3, 6, 9 and 12 in a 12-layer transformer, with results shown in Table 14. We found that adding memory to the middle of the layer stack will obtain the best result, whereas adding memory to layers either too close to the input or to the output obtained less gains.

Table 14: Different layer index.

Layer index	Perplexity
3	2.40
6	2.36
9	2.37
12	2.43

**Number of neighbors** We studied the effects of the number of neighbors we retrieve from memory, with results shown in Table 15. We found that even with 32 number of neighbors, we can already obtain a comparable results with 128 or 256 neighbors.

Table 15: Number of neighbors.

Number of neighbors	Perplexity
32	2.38
128	2.37
256	2.37

**Random seeds** We measured the statistical significant of the results reported. We did 3 runs with 3 random seeds for Transformer XL of size 512, and also a memorizing transformer with memory size 8192. We measured the standard deviation of perplexities after 500K steps of training, shown in Table 16. We saw the standard deviation between different runs of the same experiment appears to be much smaller than the gap between different models.

Table 16: Random seeds.

Models	Perplexity
Transformer XL	$2.67 \pm 0.01$
Memorizing Transformer	$2.37 \pm 0.005$

## B WHAT DOES THE MODEL RETRIEVE FROM MEMORY?

**Retrieving citation names** On arXiv math, several examples are shown in Table [17] which includes both the retrieved token and its surrounding context. We observe that many of the gains in cross-entropy loss took place when trying to predict the name of bibitems, citations, or references, by looking up the references and citations used previously in the paper. Such lookups usually span over the entire paper, which is much longer than 8192 tokens, providing a plausible explanation for the gain beyond memory size of 8192.

Table 17: The table shows several examples of which tokens were retrieved during language modelling of arXiv math dataset. The model is retrieving names of the references from previous passages.

Query index	Input	Target	Surrounding context	Retrieved index	Retrieved surrounding context
20389	Mon	thus	bibitem{ ComtetMonthusYor }	2208	Brownian motion \cite{ ComtetMonthusYor }
16623	cha	kra	\cite{ chakrabarti }.	4677	\sim 1.2 of \cite{ chakrabarti }
14747	as	d	\eqref{ asdfg } which	3365	begin{equation} \n \label{ asdfg .1 }

**Retrieving function names from the codebase** As with the arXiv papers, we also studied which tokens the model retrieved from memory. As might be expected, the model is often looking up the names of functions, and variables, as shown in Table [18]

Table 18: Examples of memory retrieval in the Github dataset. The model looks up how functions are used elsewhere in the repository.

Query index	Input	Target	Surrounding context	Retrieved index	Retrieved surrounding context
23837	Fo	nte	menu_play-> setarFonte	14607	menu_load-> setarFonte
23825	,	35	hscreen/2-50, 50, 200, 35 );	14599	20, y+40, 200, 35 )
14546	->	adi	panel-> adicionaComponente	5205	panel-> adicionaComponente

## B.1 MORE RETRIEVING EXAMPLES IN FORMAL THEOREM PROVING CORPUS

**Example 1**

- Input token index: 64604
- Input token: “\_”
- Target token: “pair”
- Surrounding context: )) by (simp add: Fourier\_sum\_limit\_pair [OF f, symmetric] Fourier’
- Name needs to be predicted: Fourier\_sum\_limit\_pair
- Retrieved token: “Four”
- Retrieved token index: 64412
- Retrieved context: 2 \* n. Fourier\_coefficient f k \* trigonometric\_set k t)
- Definition of the name:

```
lemma Fourier_sum_limit_pair:
assumes "f absolutely_integrable_on {-pi..pi}"
shows "(λn. ∑k≤2 * n. Fourier_coefficient f k * trigonometric_set k t) ⟶ l
    ⟷ (λn. ∑k≤n. Fourier_coefficient f k * trigonometric_set k t) ⟶ l"
(is "?lhs = ?rhs")
```

Figure 19: Definition of Fourier\_sum\_limit\_pair.

**Example 2**

- Input token index: 46175
- Input token: “tri”
- Target token: “gon”
- Surrounding context: <le>n. a k \* trigonometric\_set k x)
- Name needs to be predicted: orthonormal\_system\_trigonometric\_set
- Retrieved token: “gon”
- Retrieved token index: 35457
- Retrieved context: lemma orthonormal\_system\_trigonometric\_set:\n "orthonormal\_system
- Definition of the name:

```
lemma orthonormal_system_trigonometric_set:
"orthonormal_system {-pi..pi} trigonometric_set"
```

Figure 20: Definition of orthonormal\_system\_trigonometric\_set.

**Example 3**

- Input token index: 49760
- Input token: “sum”
- Target token: “m”
- Surrounding context: nusing Fourier\_series\_square\_summable [OF assms, of’
- Name needs to be predicted: Fourier\_series\_square\_summable
- Retrieved token: “sum”
- Retrieved token index: 35457
- Retrieved context: lemma Fourier\_series\_square\_summable\n assumes:
- Definition of the name:

```
lemma Fourier_series_square_summable:
  assumes os: "orthonormal_system S w" and w: " $\bigwedge i. (w_i)$  square_integrable S"
    and f: "f square_integrable S"
  shows "summable (confine ( $\bigwedge i. (\text{orthonormal_coeff } S w f i)^2$ ) I)"
```

Figure 21: Definition of Fourier\_series\_square\_summable.

**Example 4**

- Input token index: 49697
- Input token: “\_”
- Target token: “system”
- Surrounding context: lemma Riemann\_lebesgue\_square\_integrable:
 nassumes "orthonormal\_system S w"
- Name needs to be predicted: orthonormal\_system
- Retrieved token: “system”
- Retrieved token index: 28052
- Retrieved context: definition orthonormal\_system :: "'a::euclidean'
- Definition of the name:

```
definition orthonormal_system :: "'a::euclidean_space set  $\Rightarrow$  ('b  $\Rightarrow$  'a  $\Rightarrow$  real)  $\Rightarrow$  bool"
  where "orthonormal_system S w  $\equiv$   $\forall m n. \text{l2product } S (w m) (w n) = (\text{if } m = n \text{ then } 1 \text{ else } 0)$ "
```

Figure 22: Definition of orthonormal\_system.

**Example 5**

- Input token index: 34817
- Input token: “.”
- Target token: “b”
- Surrounding context: shows "integrable (lebesgue\_on {a..b})
- Retrieved token 1: “.”
- Retrieved token index 1: 2416
- Retrieved context 1: lebesgue\_on {a..b}) f i
- Retrieved token 2: “-”
- Retrieved token index 2: 2445
- Retrieved context 2: (lebesgue\_on {a-c..b-c}) (
- Retrieved token 3: “-”
- Retrieved token index 3: 6479
- Retrieved context 3: (lebesgue\_on {-pi..pi}) (

**Example 6**

- Input token index: 49759
- Input token: “\_”
- Target token: “sum”
- Surrounding context: 0"\n using Fourier\_series\_square\_summable [OF assms
- Retrieved token 1: “set”
- Retrieved token index 1: 35044
- Retrieved context 1: definition trigonometric\_set :: "nat \<Rightarrow>
- Retrieved token 2: “ier”
- Retrieved token index 2: 47272
- Retrieved context 2: definition Fourier\_coefficient\nwhere
- Retrieved token 3: “ine”
- Retrieved token index 3: 18160
- Retrieved context 3: lemma Schwartz\_inequality\_strong:\nassumes “f”
- Retrieved token 4: “system”
- Retrieved token index 4: 28052
- Retrieved context 4: definition orthonormal\_system :: “\a:euclidean”
- Retrieved token 5: “<”
- Retrieved token index 5: 47241
- Retrieved context 5: subsection\<open>Convergence wrt the L'
- Retrieved token 6: “n”
- Retrieved token index 6: 40835
- Retrieved context 6: \n subsection\<open>A bit of extra’

# Training Language Models with Memory Augmentation

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

<sup>†</sup>Princeton University

{zzhong, danqic}@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 suboptimal 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

<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>.

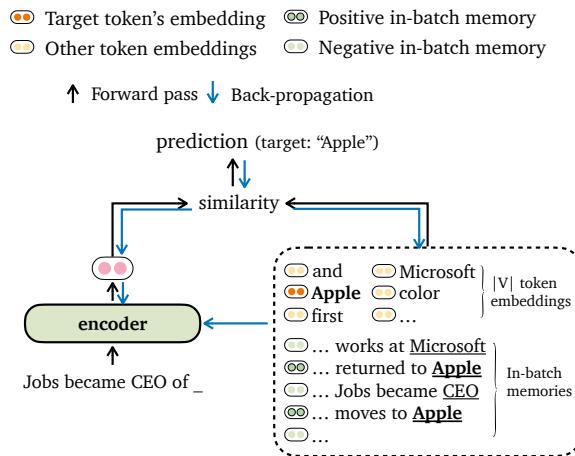


Figure 1: An illustration of our training objective. Our objective aligns the hidden representation with both token embeddings and a set of in-batch contextualized representations that are constructed during training.

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 (**T**raining with **I**n-batch **M**emories)<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>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

<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).

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>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.

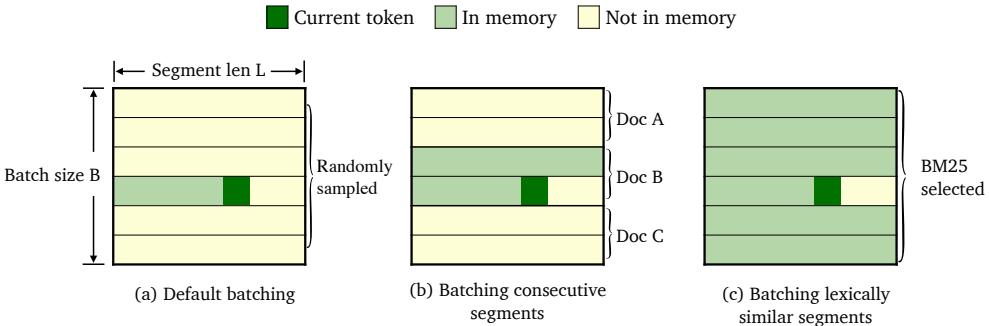


Figure 2: We present several data batching methods and memory construction strategies, in order to adapt to different sets of testing memories. (a) default batching: all the segments are randomly drawn from the training corpus (§4.1); (b) we batch consecutive segments from the same document ( $m > 1$ ) 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 (↓)	Test (↓)	Speed (↑)
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 ±0.44	3.6k t/s
★ TRIMELM	247M	17.10	17.76 ±0.94	3.6k t/s
★ TRIMELM <sub>long</sub>	247M	<b>17.01</b>	<b>17.64 ±1.06</b>	3.6k t/s
kNN-LM (our run)	247M	16.40	16.37	300 t/s
+ continuous cache	247M	16.23	16.23 ±0.14	300 t/s
★ TRIMELM <sub>ext</sub> (w/o $\mathcal{M}_{\text{long}}$ )	247M	15.62	15.55 ±0.82	300 t/s
★ TRIMELM <sub>ext</sub>	247M	<b>15.51</b>	<b>15.41 ±0.94</b>	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 ±0.33	50 t/s
★ TRIMELM <sub>ext</sub> <sup>†</sup>	247M	<b>15.40</b>	<b>15.37 ±0.75</b>	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-

resentations 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 TRIMELM<sub>ext</sub>.

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 TRIMELM<sub>long</sub> and TRIMELM<sub>ext</sub>, we tune the number of segments used in  $\mathcal{M}_{\text{long}}$  on the development set during evaluation. Our TRIMELM<sub>ext</sub> 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 kNLLM (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 (↓)	Test (↓)
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 $\downarrow 2.28$
★ TRIMELM	24.45	25.60 $\downarrow 0.27$
★ TRIMELM <sub>long</sub>	<b>21.76</b>	<b>22.66</b> $\downarrow 3.21$

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 (↓)	Test (↓)
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 $\downarrow 0.01$
★ TRIMELM	38M	1.14	1.12 $\downarrow 0.04$
★ TRIMELM <sub>long</sub>	38M	<b>1.08</b>	<b>1.05</b> $\downarrow 0.11$
SRU++ ( $L = 512$ )	42M	1.05	1.03
★ TRIMELM <sub>long</sub>	42M	<b>1.03</b>	<b>1.01</b> $\downarrow 0.02$
SRU++ ( $L = 2,048$ )	42M	1.02	0.99
★ TRIMELM <sub>long</sub>	42M	<b>1.00</b>	<b>0.98</b> $\downarrow 0.01$

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 (↓)	Test (↓)
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 (↑)
Transformer enc-dec	32.58
kNN-MT	33.15 ±0.57
★ TRIMEMT <sub>ext</sub>	<b>33.73</b> ±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		
	$\mathcal{M}_{\text{local}}$ , $\mathcal{M}_{\text{local}}$	$\mathcal{M}_{\text{local}}$ , $\mathcal{M}_{\text{long}}$	$\mathcal{M}_{\text{long}}, \mathcal{M}_{\text{ext}}$
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-

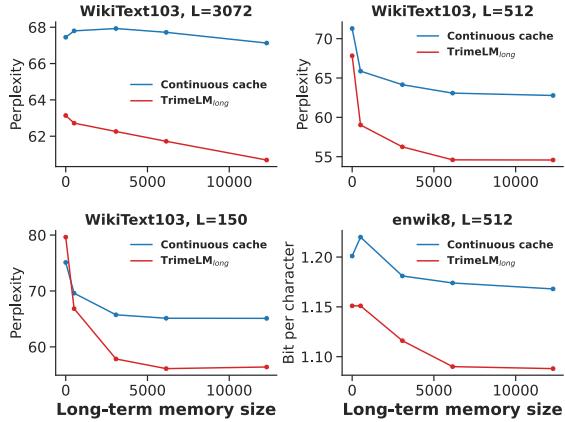


Figure 3: Performance of TRIMELM<sub>long</sub> (with different segment lengths  $L$ ) and continuous cache (adapted to long-term memory) on the WIKITEXT-103 (7M models) and ENWIK8 development sets.

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; Chormanski 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 $\times$  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, Jinhyuk Lee, and the members of Princeton NLP group for helping with proofreading and providing valuable feedback. This research is partially supported by the James Mi \*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 Weting, and William Cohen. 2022. Augmenting

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

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuhui Chen, Christopher De-wan, 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
         $| \quad c \leftarrow \text{random\_sample}(S);$ 
    end
     $l.append(c);$ 
     $S.remove(c);$ 
     $n \leftarrow \text{None};$ 
    for  $c'$  in  $\text{SimSeg}(I, c, k)$  do
        if  $c'$  in  $S$  then
             $| \quad n \leftarrow c';$ 
            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

	<b>Train</b>	<b>Dev</b>	<b>Test</b>	$ \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			ENWIKE8	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 (↓)	Test (↓)
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 ↓28.97
★ TRIMELM <sub>long</sub>	60.71	60.10 ↓23.56
★ TRIMELM <sub>ext</sub>	<b>41.50</b>	<b>42.36 ↓41.30</b>

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 (↓)
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.

---

# Long Range Language Modeling via Gated State Spaces

---

Harsh Mehta<sup>1\*</sup>Ankit Gupta<sup>2</sup>Ashok Cutkosky<sup>3</sup>Behnam Neyshabur<sup>1</sup>

## Abstract

State space models have shown to be effective at modeling long range dependencies, specially on sequence classification tasks. In this work we focus on autoregressive sequence modeling over English books, Github source code and ArXiv mathematics articles. Based on recent developments around the effectiveness of gated activation functions, we propose a new layer named *Gated State Space* (GSS) and show that it trains significantly faster than the diagonal version of S4 (i.e. DSS) on TPUs, is fairly competitive with several well-tuned Transformer-based baselines and exhibits zero-shot generalization to longer inputs while being straightforward to implement. Finally, we show that leveraging self-attention to model local dependencies improves the performance of GSS even further.

## 1 Introduction

Modeling long range dependencies on sequential data is a crucial step towards closing the gap with human-level performance on many tasks. Attention based models like Transformer [Vaswani et al., 2017] have proven to be a strong choice of backbone architecture for a considerable number of tasks across modalities and scale [Devlin et al., 2019, Brown et al., 2020, Dosovitskiy et al., 2021]. Vanilla Multi-Head-Attention famously incurs  $\Omega(L^2)$  penalty in modeling a sequence of length  $L$ . This is prohibitive at best for tasks where the model is required to capture long range dependencies from various parts of the input. Over the years, a variety of improvements have been proposed to alleviate this quadratic complexity (cf. [Tay et al., 2020]).

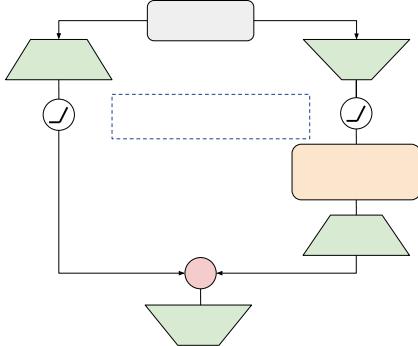
On a somewhat orthogonal direction, attention-free models based on state spaces, such as S4 [Gu et al., 2022a] and DSS [Gupta et al., 2022], have shown remarkable improvements on Long Range Arena (LRA) [Tay et al., 2021], a benchmark designed with long range modeling as its focus and consists of diverse tasks with 1k-16k sequence length across modalities. These models require careful initialization, originally borrowing ideas from the theory of HiPPO matrices [Voelker et al., 2019, Gu et al., 2020], to achieve good results on LRA.

In this work, we explore and extend the use of state space models by focusing solely on the task of autoregressive sequence modeling [Brown et al., 2020, Rae et al., 2021, Chowdhery et al., 2022, Zhang et al., 2022, Hoffmann et al., 2022, Srivastava et al., 2022]. Several key properties endowed by the state space model family makes it particularly attractive, to at least fully explore it, in the context of language modeling. First, it reduces the  $\Omega(L^2)$  complexity on input sequence length to  $O(L \log L)$ . This complexity results from the use of Fast Fourier Transform (FFT) [Cooley and Tukey, 1965] for performing convolutions. We will describe this in detail in later sections. Second, the state space model is fully parallelizable in the length dimension. This is an arguably subtle but an important property at training time. Note that transformers are also fully parallelizable, a worthy advantage over traditional RNNs for modeling sequences, which otherwise incurs only an  $O(L)$  penalty. While this parallelism is useful at training time, it may also be a curse at inference time

---

\*<sup>1</sup>Google Research, <sup>2</sup>Tel Aviv University, <sup>3</sup>Boston University.

harshm@google.com, ankitgupta.iitkanpur@gmail.com, ashok@cutkosky.com, neyshabur@google.com.



```

def gss(x, F=4096, L=4096, E=1024, H=256):
    shortcut, x = x, norm(x)
    v = dense(x, F, activation='gelu')
    u = dense(x, H, activation='gelu')
    y = dss(u, H, L)
    # yh1, ..., yhL are linear in uh1, ..., uhL
    uc = dense(y, F)
    o = dense(uc * v, E)
    return o + shortcut

```

Figure 1: (a) Our proposed Gated State Space (GSS) layer, (b) Pseudocode for GSS (full implementation in §A.2).

where decoding every token requires attending to the whole past. The ideal model is parallelizable at training time but incurs a small constant cost (per decoded token) at inference time. This brings us to the final point. Due to the inherent convolution-recurrence equivalence of the state space model, it can be made to accumulate state and unroll like an RNN at inference time without any approximations.

Despite these attractive properties, we found that current state space models (such as S4, DSS) run slower than we expected at training time on TPUs, our accelerator of choice. We take this opportunity to modify the architecture to reduce dimensionality of specific operations which we found to be bottlenecks. Our proposed changes borrow from a well-supported empirical observation around the effectiveness of gating units [Shazeer, 2020]. Specifically, Hua et al. [2022] observed that replacing the typical Feed-Forward layer in the Transformer with gating units allows for a reduced dimensionality when mixing tokens along the length dimension using self-attention. We extend the use of gating units to state space model family and observe that, even in our context, the use of gating units allows for a reduction in dimensionality when performing FFT operations, which we observed to be the main bottleneck behind slow training. Furthermore, somewhat contrary to observations made by S4 and DSS authors, we found the performance of the model on language modeling tasks to be much less sensitive to initialization. We found that only the scale and structural aspects of initialization of state space variables were important and not the exact values. We were able to successfully train the model while initializing the state space variables randomly. This departs significantly, at least in understanding, from the reliance of the design on the theory of HiPPO matrices, which led the S4 model to employ several numerical linear algebra tricks to make it work. Combining both of these contributions, we propose a layer named Gated State Space (GSS) (Figure 1), which we empirically verified to be 2-3× faster than DSS while keeping the perplexity on several language modeling benchmarks (Table 1).

Going one step further, we also perform an apples-to-apples comparison with well-tuned and performant baselines reported in Block Recurrent Transformers [Hutchins et al., 2022], on several long range language modeling benchmarks over modalities such as English books, raw source code from Github and LaTeX source of ArXiv mathematics articles. As detailed in Table 2, while our GSS model currently lags behind on some tasks when compared in the fixed-parameter setting, it is fairly competitive in the *fixed-compute* setting where we measure compute as the exact amount of TPUv4 hours spent on a training run and serves as a fairly accurate proxy to the realistic cost of training that model. Furthermore, we also experimented with a hybrid model in which we sparingly interleave Transformer layers (having local attention) in a GSS stack to allow for a richer modeling of short range interactions. To our delight, this further improves performance at (roughly) no extra training cost, both in terms of parameters and compute.

While in our experiments we train on sequences of length at most 4k, we evaluated our GSS variants on a wide range of sequence lengths upto 65k and found consistent generalization to longer inputs. Not only the performance doesn't degrade as the sequence length is increased but it gets significantly better, suggesting that GSS is effective at utilizing the extra context even though it was not trained with that much amount of context.

At inference time, state space models including GSS are fairly efficient since decoding can happen in recurrent mode (as much as 60× better in the case of S4 [Gu et al., 2022a]). Though, the hybrid model which also uses local attention complicates this advantage a bit.

In summary, we propose GSS, an alternative to S4 and DSS which trains 2-3 $\times$  faster, is simple to implement and fairly competitive with well-tuned Transformer-based baselines on several long range language modeling benchmarks.

## 2 Related Work

In recent years, attention-based models have emerged as a dominant technique for sequence modeling, achieving remarkable improvements in a wide range of tasks, starting in NLP [Vaswani et al., 2017, Devlin et al., 2019, Radford et al., 2019, Liu et al., 2019], then moving to other classical machine learning areas such as computer vision [Dosovitskiy et al., 2021] and now to the physical sciences [Avsec et al., 2021, Jumper et al., 2021]. In brief, an attention layer takes as input three matrices  $K, Q, V$  in  $\mathbb{R}^{L \times d}$ , which should usually be thought of as length  $L$  lists of  $d$ -dimensional vectors. The attention layer then outputs  $Y = \sigma(QK^\top)V \in \mathbb{R}^{L \times d}$  where  $\sigma$  indicates a row-wise softmax operation. In the popular self-attention variant,  $K, Q$  and  $V$  are themselves learned functions (usually simple linear transformations) of a single input sequence  $X = (x_1, \dots, x_L)^\top \in \mathbb{R}^{L \times d}$ . Intuitively, one might imagine that this mechanism can model long range dependencies because the “computational path” between any row  $x_i$  and any row  $y_j$  is very short: more formally, this model avoids the “vanishing gradients” issue that plagues RNNs, in which the partial derivative  $\frac{\partial y_L}{\partial x_1}$  decays exponentially in  $L$ . Unfortunately, this requires  $O(L^2)$  time and space due to the need to construct the matrix  $QK^\top \in \mathbb{R}^{L \times L}$ , which places a limitation on how large  $L$  can be.

The surge in popularity of attention has engendered a corresponding surge in interest on methods for increasing the context length  $L$  while controlling the  $O(L^2)$  computational cost. Broadly speaking, most approaches fall into two camps: those that attempt to “linearize” attention, and those that sparsify the attention matrix  $QK^\top$ . The first camp exploits the fact that if the softmax is removed, we can re-write attention as  $Q(K^\top V)$ , where now  $K^\top V \in \mathbb{R}^{d \times d}$  and so the whole operation is only linear in  $L$  rather than quadratic. Thus, if we could replace the softmax with a different operation that allowed some version of this rearrangement, it would be possible to speed up the attention layer. This idea is the underlying principle behind methods such as the Performer [Choromanski et al., 2021], Linear Attention [Katharopoulos et al., 2020], Random Feature Attention [Peng et al., 2021] or cosFormer [Qin et al., 2022]. In the other camp, one employs the simple expedient of simply not computing the entire matrix  $QK^\top$ , usually by employing some kind of sparsification strategy to only compute some of the elements of this matrix. For example, BigBird [Zaheer et al., 2020], GMAT [Gupta and Berant, 2020], Longformer [Beltagy et al., 2020], and Blockwise self-attention [Qiu et al., 2020] all choose particular sparsification patterns that are intuitively likely to capture “important” interactions, while the Reformer [Kitaev et al., 2020] uses locality-sensitive hashing to dynamically adjust the sparsification pattern. Most recently, Hawthorne et al. [2022] compress the history in a small number of latents for long range autoregressive tasks. In essence, these approaches can be viewed as a trade-off between performance, and computation time (or memory).

Despite its current empirical success, the attention mechanism is not the only approach for modeling sequence data, nor even necessarily the most natural. For example, a classical *recursive* or *state space* layer operates on an input  $(x_1, \dots, x_L)^\top \in \mathbb{R}^{d \times L}$  by defining a sequence of “states”  $s_{t+1} = T(s_t, x_{t+1})$  and returning the output sequence  $y_t = E(s_t)$  where  $T$  and  $E$  are learned “transition” and “emission” functions. Assuming  $T$  and  $E$  can be computed efficiently, this requires only  $O(L)$  time. Note however, that the  $O(L)$  theoretical complexity is complicated by the fact that these models appear to be inherently serial, while the  $O(L^2)$  matrix operations in attention are much more easily parallelizable. State space models have additional attractive properties: they naturally capture our physical intuition that the universe is described by some appropriate state, and the output  $y_t$  can be theoretically influenced by every single prior input, rather than requiring the model to commit to some context window of fixed length  $L$ . Through the years, many different possibilities for  $T$  and  $E$  have appeared, such as the simple RNN, which sets  $T$  and  $E$  to both be MLPs, or more complicated LSTM layer [Hochreiter and Schmidhuber, 1997]. Nevertheless, the performance of state space models has in many cases been quickly surpassed by attention.

There have been several recent efforts to recapture the favorable properties of state space models while maintaining or improving upon the performance of attention models. For example, the Transformer-

XL [Dai et al., 2019] modifies attention to incorporate state via a sliding-window style approach, while the Block-Recurrent Transformer [Hutchins et al., 2022] directly parameterized the  $T$  and  $E$  functions using attention layers. Further, alternative models based on linear dynamical systems have shown great promise [Gu et al., 2022a, Gupta et al., 2022, Gu et al., 2022b]. Rather than attempting to “fix” attention, these are classical state space models where both  $T$  and  $E$  are linear. An important insight in these works is to recast the model as a convolution with a very large kernel, and then leverage the convolution theorem to compute the sequence  $y_t$  in  $O(L \log(L))$  time using Fast Fourier Transform [Cooley and Tukey, 1965]. While seemingly worse than  $O(L)$ , this operation is easily parallelizable, and so in practice is significantly faster. Moreover, in a situation in which the  $\log(L)$  factor is onerous, one may still fall back to the serial  $O(L)$  algorithm. Despite their apparent simplicity, these models have demonstrated remarkable success in tasks that require integrating information from distant parts of the input: the key innovation is that careful initialization and parameterization of the linear maps is required in order to train the models effectively. Our approach will build on these works.

### 3 Method

We start by reviewing the necessary background on state spaces required to fully describe our model (§3.1). We then formally define our GSS model in §3.2 and the GSS-Transformer-Hybrid in §3.3.

#### 3.1 State Space Preliminaries

While in this work we are interested in modeling sequences of vectors, let us first review how state space models define a sequence-to-sequence map for 1-D sequences.

**State Spaces** A discretized state space, parameterized by a state matrix  $A \in \mathbb{R}^{N \times N}$ , vectors  $B \in \mathbb{R}^{N \times 1}$ ,  $C \in \mathbb{R}^{1 \times N}$  and a sample time  $\Delta \in \mathbb{R}_{>0}$  defines a sequence-to-sequence map from input  $(u_0, \dots, u_{L-1}) = u \in \mathbb{R}^L$  to output  $(y_0, \dots, y_{L-1}) = y \in \mathbb{R}^L$  via the recurrence<sup>2</sup>,

$$\begin{aligned} x_k &= \bar{A}x_{k-1} + \bar{B}u_k , \quad y_k = \bar{C}x_k \\ \bar{A} &= e^{A\Delta} , \quad \bar{B} = (\bar{A} - I)A^{-1}B , \quad \bar{C} = C . \end{aligned} \tag{1}$$

Assuming  $x_{-1} = 0$  for simplicity, the above recurrence can be explicitly unrolled as

$$y_k = \sum_{j=0}^k \bar{C}\bar{A}^j\bar{B} \cdot u_{k-j} . \tag{2}$$

For convenience, the scalars  $\bar{C}\bar{A}^k\bar{B}$  are gathered to define the SSM kernel  $\bar{K} \in \mathbb{R}^L$  as

$$\bar{K} = (\bar{C}\bar{B}, \bar{C}\bar{A}\bar{B}, \dots, \bar{C}\bar{A}^{L-1}\bar{B}) = (Ce^{A \cdot k \Delta} (e^{A\Delta} - I)A^{-1}B)_{0 \leq k < L} , \tag{3}$$

where the last equality follows by substituting the values of  $\bar{A}$ ,  $\bar{B}$ ,  $\bar{C}$  from Equation 1. Hence,

$$y_k = \sum_{j=0}^k \bar{K}_j \cdot u_{k-j} . \tag{4}$$

where  $\bar{K}_j$  denotes the value of the kernel at position  $j$ . Given an input sequence  $u \in \mathbb{R}^L$ , it is possible to compute the output  $y \in \mathbb{R}^L$  sequentially via the recurrence in Equation 1. While this property is highly desirable for autoregressive decoding, a sequential computation is prohibitively slow to train with long inputs and, instead, Equation 4 can be used to compute all elements of  $y$  in parallel, provided we have already computed  $\bar{K}$ .

---

<sup>2</sup>Discretization used in Equation 1 relies on zero order hold (ZOH) which assumes a constant value between sampled times. We also experimented with other discretization methods such as bilinear transform but did not see much difference in performance.

**Computing  $y$  from  $u$  and  $\bar{K}$  is easy.** Given an input sequence  $u \in \mathbb{R}^L$  and the SSM kernel  $\bar{K} \in \mathbb{R}^L$ , naively using Equation 4 for computing  $y$  would require  $O(L^2)$  multiplications. Fortunately, this can be done much more efficiently by observing that for the univariate polynomials

$$\bar{K}(z) = \sum_{i=0}^{L-1} \bar{K}_i z^i \quad \text{and} \quad u(z) = \sum_{i=0}^{L-1} u_i z^i,$$

$y_k$  is the coefficient of  $z^k$  in the polynomial  $\bar{K}(z) \cdot u(z)$ , i.e. all  $y_k$ 's can be computed simultaneously by multiplying two degree  $L - 1$  polynomials. It is well-known that this can be done in  $O(L \log(L))$  time via Fast Fourier Transform (FFT) [Cormen et al., 2009]. We denote this fast computation of Equation 4 via the discrete convolution as

$$y = \bar{K} *_c u. \quad (5)$$

**Diagonal State Spaces** The challenging part is computing  $\bar{K}$  itself as it involves computing  $L$  distinct matrix powers (Equation 3). Gupta et al. [2022] observed that the state matrix  $A$  can be assumed to be diagonal without loss in performance, thereby allowing a straightforward computation of  $\bar{K}$ . Their DSS<sub>EXP</sub> model (DSS from hereon) assumes  $A$  to be a diagonal matrix  $\text{diag}(\lambda_1, \dots, \lambda_N)$  and assumes  $B = (1)_{1 \leq i \leq N}$ . DSS has parameters  $\Lambda_{\text{re}}, \Lambda_{\text{im}} \in \mathbb{R}^N$ ,  $C \in \mathbb{C}^N$  and  $\Delta_{\log} \in \mathbb{R}$ . The diagonal of  $A$  (i.e.  $(\lambda_1, \dots, \lambda_N)$ ) is computed as  $- \text{elementwise-exp}(\Lambda_{\text{re}}) + i \cdot \Lambda_{\text{im}}$  where  $i = \sqrt{-1}$  and  $\Delta$  is computed as  $\exp(\Delta_{\log}) \in \mathbb{R}_{>0}$ . For this parameterization, the kernel (Equation 3) can be computed as a matrix-vector product

$$\bar{K} = (C * ((e^{\lambda_i \Delta} - 1) / \lambda_i)_{1 \leq i \leq N})_{1 \times N} \cdot \text{elementwise-exp}(P_{N \times L}) \quad (6)$$

where  $P_{i,k} = \lambda_i k \Delta$  and  $*$  denotes elementwise multiplication.

This formally defines a *linear* sequence-to-sequence map for 1-D sequences. In the case of sequences of  $H$ -dimensional vectors, state space models are applied individually on the  $H$  features as follows.

Each DSS layer receives a length- $L$  sequence  $u$  of  $H$ -dimensional vectors as input, i.e.,  $u \in \mathbb{R}^{H \times L}$ , and produces an output  $y \in \mathbb{R}^{H \times L}$ . The parameters of the layer are  $\Lambda_{\text{re}}, \Lambda_{\text{im}} \in \mathbb{R}^N$ ,  $\Delta_{\log} \in \mathbb{R}^H$  and  $C \in \mathbb{C}^{H \times N}$ . For each coordinate  $h = 1, \dots, H$ , a state space kernel  $\bar{K}_h \in \mathbb{R}^L$  is computed for  $\Lambda_{\text{re}}, \Lambda_{\text{im}}, (\Delta_{\log})_h, C_h$  via Equation 6. The output  $y_h \in \mathbb{R}^L$  for coordinate  $h$  is computed from  $u_h \in \mathbb{R}^L$  and  $\bar{K}_h$  using Equation 5.

For batch size  $B$ , sequence length  $L$  and hidden size  $H$ , the DSS layer requires  $O(NHL)$  time to compute the kernels, and  $O(BHL \log(L))$  time for the discrete convolution. In S4 and DSS, the authors suggest to use  $N = 64$  as default and for this choice of  $N$ , the time taken to compute the kernels becomes an important factor, specially for small batches.

### 3.2 GSS Layer

Armed with the tools and techniques from the state space literature, we now turn to the main contribution of our work and describe our GSS layer in detail (Figure 1).

We build on the idea of the recently introduced Gated Attention Unit (GAU) [Hua et al., 2022] and replace the  $\Omega(L^2)$  attention used in GAU by a further simplified DSS layer (§3.1). Gating allows our model to be contextualized over a reduced dimensionality and the use of state spaces provides it with superior contextualizing abilities, while enjoying  $O(L \log L)$  complexity out of the box.

Concretely, our *Gated State Space* (“GSS”) layer maps a sequence  $X \in \mathbb{R}^{L \times E}$  to an output  $O \in \mathbb{R}^{L \times E}$  as

$$U = \phi(XW_1) \in \mathbb{R}^{L \times H} \quad V = \phi(XW_2) \in \mathbb{R}^{L \times F} \quad (7)$$

$$Y = \text{DSS}(U) \in \mathbb{R}^{L \times H} \quad U_{\text{context}} = YW_3 \in \mathbb{R}^{L \times F} \quad (8)$$

$$O = (U_{\text{context}} * V)W_4 \in \mathbb{R}^{L \times E} \quad (9)$$

where  $*$  denotes elementwise multiplication and  $\phi$  denotes GELU [Hendrycks and Gimpel, 2016]. Note that, contrary to Hua et al. [2022], in our experiments, we did not observe much benefit from using RELU<sup>2</sup> or Swish activations instead of GELU.

In Equation 8, a contextualized representation of  $U$  is formed via DSS in  $O(NHL) + O(BHL \log L)$  time. In most of our experiments we used  $H = E/4 = 256$ ,  $N = 512$  and  $F = 4096$ . Similar to GAU, gating allows us to perform a weaker contextualization using the state space layer by reducing the dimension of the input to DSS, by  $4\times$  in our case. This offers a much needed increase in throughput since we observed that FFT ops were the main bottleneck (measured on TPUs). As shown in Table 1, this provides more than  $3\times$  speedups over the vanilla DSS block described in §3.1.

**Simplified DSS used in GSS layer** In Equation 8, instead of directly using DSS as described in §3.1, we made several key simplifications based on our exploratory results.

As described in §3.1, S4 and DSS train a separate  $\Delta$  for each of the  $H$  coordinates resulting in the creation of  $H$  separate  $P$  matrices in Equation 6. In the DSS layer used in GSS, we decided to eliminate the use of  $\Delta$  by fixing it as 1. This reduces the compute required for creating the kernels and makes the kernel computation simpler.

Secondly, we found that randomly initializing the parameters corresponding to  $\Lambda$  works just as well as the Skew-Hippo initialization suggested in [Gupta et al., 2022]. In practice, we parametrize both real and imaginary parts of  $\Lambda$  in log space (§A.2). The effectiveness of random initialization is in contrast to the findings of Gu et al. [2022a] and Gupta et al. [2022] who reported their respective initializations of the state space parameters to be critical to the model performance. We do however note that the experiments in our setting of large-scale language modeling are conducted on orders of magnitude larger scale of compute than what is used in the tasks considered in these works. Moreover, the data modalities we consider in this work are different from the tasks considered in these works (e.g. Long Range Arena) which are specifically designed to stress test long-range reasoning.

### 3.3 GSS-Transformer-Hybrid

Conceptually, GSS looks fairly different from the current workhorse of machine learning; the Transformer architecture. Given this, it is not immediately clear if one is decidedly better than the other or if they each provide some orthogonal benefits. If its the latter, one might wonder if there are synergies between these architectures which can be exploited to create a hybrid model which is stronger than either one of them individually. To that end, we also consider a conceptually simple hybrid between GSS and Transformer where we sparingly interleave traditional Transformer blocks with GSS layers. Despite its glaring simplicity, as shown in Table 2, we observed that it shows consistent and significant improvements on all tasks.

**Chunking long inputs** In all our experiments we used sequence lengths large enough to be prohibitive for traditional Transformer layers. To get around this restriction at the Transformer layers used in our hybrid model, we chunk their inputs into non-overlapping chunks of length 512 and run the Transformer layer on each of them independently. While the GSS layers are apt at modeling both short and longer range interactions, the interleaved Transformer layers can potentially allow for a richer modeling of short range interactions.

## 4 Results

We conduct experiments with GSS on 4 different datasets, LM1B, PG19, ArXiv and Github, each of them varying qualitatively with another in terms of modality and average document length.

**LM1B** is a standard and reasonably big dataset (1B tokens) where each training example consists of a short sentence [Chelba et al., 2014]. This is different from rest of the datasets which consists of a much larger sequence of tokens per example. Although, our primary goal is to measure GSS’s ability to capture long range dependencies, we include results on LM1B benchmark and compare with vanilla Transformer baseline, which is hard to do for larger sequence lengths.

**PG19** dataset is constructed from extracting a large collection of full-length books from Project Gutenberg [Rae et al., 2020]. All extracted books were written before 1919 and only contain tokens from the English language. Over the years PG-19 has become a standard benchmark for measuring progress on long range dependency modeling over text.

**ArXiv Math** dataset was recently collected by Wu et al. [2022] and contains LaTeX source for articles focusing on Mathematics. Even though articles are typically shorter than full-length books,

Dataset	Model	Params	Throughput (steps/sec)	Eval Sequence Length			
				512	4k	16k	65k
LM1B	Transformer	182M	6.6	13.51			
	DSS	188M	1.8	13.59			
	GSS	190M	5.6	13.26			
PG-19	DSS	209M	1.8	14.51	13.53	692.1	OOM
	GSS	192M	5.3	14.01	12.84	12.94	12.47
Arxiv	DSS	209M	1.8	3.65	3.13	284.6	OOM
	GSS	192M	5.3	3.57	3.08	3.08	2.75
Github	DSS	209M	1.8	3.65	3.21	242.7	OOM
	GSS	192M	5.3	2.68	2.35	2.31	2.12

Table 1: Comparison of DSS and GSS models in fixed-param setting. We consistently find that GSS outperforms DSS (with hyperparameters taken from [Gupta et al. \[2022\]](#)) while being 2-3× faster on all tasks. Training sequence length is 4k, except for LM1B for which it is 512.

as in PG19, since LaTeX source can have many special characters, typical sub-piece vocabularies are not very effective and tends to produce examples of similar size, as measured by number of tokens. It is possible to train a custom sub-piece vocabulary for this dataset, but we stick with the vocabulary used by both [Wu et al. \[2022\]](#) and [Hutchins et al. \[2022\]](#) for fair comparison.

**Github** was also first collected and used by [Wu et al. \[2022\]](#). It is a corpus of raw source code collected from several Github repositories with open-source licences. The dataset contains code from several programming languages, including C, C++, Java, Python, Go and Typescript. In this case individual files can be small but code repositories are typically organized so that code can be reused across file boundaries. Thus, for every repository, all the files were concatenated to produce a single document.

We do token level modeling on all the datasets and report resulting perplexity numbers on a heldout set of examples. Perplexity numbers are obtained using teacher forcing (or parallel mode) where the correct output from the heldout set is used for decoding the next token at each position. For a fair comparison with the baselines we considered, we keep the vocabularies consistent as used by the baselines models. Specifically, we used custom trained 30k sized sentence-piece vocab for LM1B, T5 vocab with 32k tokens for PG19 [[Raffel et al., 2020](#)] and Meena vocab with 32k tokens [[Adiwardana et al., 2020](#)] for both ArXiv and Github datasets.

#### 4.1 Comparing DSS and GSS

As shown in Table 1, in our first set of results, we compare DSS and GSS models both in terms of perplexity and throughput, as measure by steps per second on all 4 datasets. Except for LM1B, all models were trained with 4k sequence length at training time. However, at evaluation time, we present results for on a large range of sequence lengths. For LM1B, we train and evaluate with sequence length of 512, since the dataset only contains short sentences, most examples comfortably fully fitting in. Despite being a (slightly) smaller model, GSS performs quite favorably as measured by perplexity while being 2-3× faster depending on the dataset.

Furthermore, we observe significant generalization over changes in sequence length on all the datasets. Not only does the performance not degrade when increasing sequence length, it actually improves quite a bit! This suggests that the model is effective at utilizing the extra context even though the model was not trained with that amount of context. Note that we used default hyperparameters suggested by [Gupta et al. \[2022\]](#) for initializing state space variables for DSS. But, for GSS, since we made several structural changes, we retuned the hyperparameters related to initialization on PG19 alone and used them for all the datasets. Since we had access to evaluation metrics for all sequence lengths, length generalization factored in for our choice of hyperparameters, which is not true for DSS. It is completely possible we see similar length generalization even with DSS if initialization hyperparameters were retuned.

Dataset	Model	Params	TPUv4 hours	Eval Sequence Length			
				Perplexity			
				512	4k	16k	65k
PG-19	Rec:fixed:skip	196M	0.8k		11.55		
	Feedback:lstm:single	196M+	0.8k+		11.31		
	Feedback:fixed:skip	196M+	0.8k+		11.24		
	<b>GSS (this work)</b>	192M	0.5k	14.01	12.84	12.94	12.47
	<b>GSS-L</b>	352M	0.8k	12.48	11.33	11.16	11.12
	<b>GSS-Hybrid-L</b>	373M	0.8k	11.45	<b>10.52</b>	10.44	10.1
Arxiv	Rec:fixed:skip	196M	0.8k		2.36		
	Feedback:lstm:single	196M+	0.8k+		<b>2.33</b>		
	Feedback:fixed:skip	196M+	0.8k+		2.36		
	GSS	192M	0.5k	3.57	3.08	3.08	2.75
	GSS-L	352M	0.8k	3.29	2.71	2.72	2.51
	GSS-Hybrid-L	373M	0.8k	2.94	2.51	18.27	125.2
Github	Rec:fixed:skip	196M	3k		2.04		
	Feedback:lstm:single	196M+	3k+		2.07		
	Feedback:fixed:skip	196M+	3k+		2.16		
	GSS	192M	0.5k	2.68	2.35	2.31	2.12
	GSS-L	352M	1.8k	2.31	1.99	2.20	2.28
	GSS-Hybrid-L	373M	1.8k	2.34	<b>1.88</b>	1.74	2.09

Table 2: Comparing GSS variants with best-performing models from Hutchins et al. [2022] in both fixed-param and fixed-compute settings. While, in the fixed-param setting, GSS currently lags behind performance of best Block Recurrent Transformer baseline, it is fairly competitive in the fixed-compute setting (as measured by total TPUv4 hours taken to complete training). For the larger model GSS-L used for fixed compute comparison, we simply double the layers from 16 to 32 keeping everything else fixed. In addition, due to the inherent recurrent view of state space model family, decoding at inference time is much faster than Transformer based baselines. For block-recurrent baselines, adding feedback increases both parameter count and training time, we stick with conservative estimates derived from Hutchins et al. [2022] for both, which we denote by ‘+'. Hutchins et al. [2022] report  $\log_2$  of perplexity which we convert to raw perplexity in this table. Param count for all the models include embedding layers as well.

**Training details** All the models were trained with  $2^{19}$  tokens per batch and 125k total training steps. We make sure to change the batch size as a function of the sequence length so that number of tokens in the batch remains the same. For example, for LM1B we set batch size to 1024 and sequence length to 512 but for rest of the datasets we use batch size of 128 and sequence length of 4k. For datasets with longer documents, we also considered increasing the sequence length even further. Intuitively, training on longer sequences would help the model learn longer range dependencies better. On the flip side, it makes the optimization a bit more challenging due to large number of correlated tokens per batch and would even likely result in overfitting. Since GSS is able to generalize beyond the length it was trained on in most cases, we found sequence length of 4k to be a reasonable middle ground for this set of experiments.

Similar to language modeling experiments in [Gu et al., 2022a], every block of DSS baseline consists of DSS layer followed by GLU [Dauphin et al., 2017] and a Feedforward layer similar to the one used in Transformer block with GELU activation [Hendrycks and Gimpel, 2016]. DSS baseline consists 12 layers and an embedding dimension of 1024. For GSS, we increased the number of layers to 16 to match the parameter count.

## 4.2 Comparison with other baselines

In this section, we turn our attention towards apples-to-apples comparison of GSS versus well-tuned baselines on these datasets. For a complete picture of the cost of training these models, we report both the number of parameters and time spent training as measured by TPUv4 hours. For baselines, we

selected the best performing models reported in [Hutchins et al., 2022] for every dataset and compare with GSS model both in fixed-param and fixed-compute settings.

**Training Details** Similar to Section 4.1, unless otherwise mentioned, all the non-baseline models were trained using 64 TPUv4 cores for 125k steps. We use batch size of 128 and 4k sequence length at training time, with a total of  $2^{19}$  tokens per batch. We increase the batch size to 256 for the Github dataset (with a token count of  $2^{20}$  per batch) since we observed a lot of noise in our metrics. This is consistent with observations made by Hutchins et al. [2022].

We used Adam optimizer [Kingma and Ba, 2015] and tuned the base learning rate over a grid of  $\in [0.0064, 0.0032, 0.0016, 0.0008]$ . We also employ linear warmup for 1k steps and cosine decay until 1e-6. We also observed better performance by using a higher than typical weight decay rate of 0.1, other than that we did not employ any additional regularization techniques, including dropout. Note that similar to [Gu et al., 2022a, Gupta et al., 2022], we used a constant learning rate of 1e-3 and set weight decay rate to 0.0 for state space parameters part of GSS Layer . In addition, we clip the gradient to norm 1.0 before passing to the optimizer. Both of these helped with certain instabilities we observed in our preliminary experiments.

**GSS models** GSS consists of 16 layers and an embedding dimension of 1024. We also consider a larger variant with 32 layers as denoted by GSS-L. For GSS-Hybrid model, we used vanilla Transformer blocks at every 4th layer starting with the 2nd layer. Since GSS layers are inherently position aware, using them for the 1st layer eschews any need of explicit position embeddings typically used with otherwise position invariant Transformer blocks. Thus, barring position aware nature of GSS layers, we don’t use any kind of explicit position embedding or bias in our models. For the Transformer blocks used in hybrid models, we use multi-head self-attention with 8 heads, each with size 128.

**Baselines** We considered 3 high-performing baselines and numbers reported in [Hutchins et al., 2022]. Block Recurrent Transformer leverages recurrence over blocks of Transformer layer to model long range dependencies. Hutchins et al. [2022] performed a comprehensive exploration of open design space of incorporating recurrence across blocks. Somewhat surprisingly, Rec:fixed:skip, which accumulates recurrent state vector as an exponential moving average over time performs better than more complicated gating designs. Another variation which performed well with Block Recurrence is the idea of adding a feedback mechanism over blocks similar to Feedback Transformer [Fan et al., 2020]. Note that feedback mechanism makes training more expensive due to additional cross-attention modules and corresponding paramaters.

**Fixed-param comparison** As shown in Table 2, we see that GSS variants come very close but not quite beat the strongest block recurrent baseline in the fixed-param setting. In this case, we are comparing GSS model which has roughly 192M parameters (including embeddings) with the baselines all of which have around 196M parameters. Even though the parameter count is fixed, we see that GSS runs faster than block recurrent baselines, likely due to the fact that all the layers can be completely parallelized unlike the the recurrence in the baselines which run in a sequential fashion.

**Fixed-compute comparison** Since the GSS model runs faster than the baselines, we also train with versions larger than GSS such that the training time (as measured by total TPUv4 hours) matches the time taken by the baselines. We simply double the number of layers from 16 to 32 to construct GSS-L. As expected, adding more parameters improves perplexity numbers on the eval set of all the datasets. Moreover, we find that the GSS-Hybrid versions of the model outperform the best baseline model on PG-19 and Github datasets. We do see significant improvements for Arxiv dataset as well but unfortunately not enough to be stronger than the baseline. We think this may be resolved by the use of a vocabulary more suited to Arxiv symbols. On the flip side, we can no longer do a fair comparison with token level perplexity if we change the vocabulary, so we stick with the vocabularies used by the baselines for this study.

**Length generalization** We train all variants of GSS with sequence length of 4k but evaluate on 4 different lengths  $l \in [512, 4k, 16k, 65k]$ . On PG-19, we see significant length generalization across the board. Not only the performance doesn’t degrade as the sequence length is increased but it gets significantly better for all model variants. On Arxiv and Github, the situation is a little more complicated. For smaller models, we still see length generalization but it tends to degrade when either the model is made bigger or the dataset has a lot of noise (as indicated by variation in perplexity metric over time). How to robustly achieve length generalization is an interesting research question

on its own and we believe one can design interventions which can lead to further improvements, which we don't explore here.

Note that block recurrent baselines, with or without feedback mechanism, process documents in a sequential fashion such that recurrent state from previous segment from the document is passed to the next segment, with backward pass truncated to the current segment. This means that, even though the segment sequence length is set to 4k, block recurrent models have (arguably weak) access to almost the entire past. Thus, perplexity comparison at sequence length 4k is slightly unfair towards GSS models since they do *not* employ such state caching mechanisms.

## Acknowledgments

We are grateful to the developers of Jax and Flax libraries. In addition, we would like to thank DeLesley Hutchins and Imanol Schlag for answering our questions, sharing the datasets and helping us in general with details of Block Recurrent Transformer paper. Finally, we are grateful to Ethan Dyer for discussions on long context modeling, Walid Krichene and John Anderson for providing feedback on an earlier draft of this paper.

## Bibliography

Daniel Adiwardana, Minh-Thang Luong, David R. So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, and Quoc V. Le. Towards a human-like open-domain chatbot. *ArXiv*, abs/2001.09977, 2020.

Žiga Avsec, Vikram Agarwal, Daniel Visentin, Joseph R Ledsam, Agnieszka Grabska-Barwinska, Kyle R Taylor, Yannis Assael, John Jumper, Pushmeet Kohli, and David R Kelley. Effective gene expression prediction from sequence by integrating long-range interactions. *Nature methods*, 18(10):1196–1203, 2021.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *ArXiv preprint*, abs/2004.05150, 2020. URL <https://arxiv.org/abs/2004.05150>.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html>.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, T. Brants, Phillip Todd Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH*, 2014.

Krzysztof Marcin Choromanski, Valerii Likhoshesterstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL <https://openreview.net/forum?id=Ua6zuk0WRH>.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek B Rao, Parker Barnes, Yi Tay, Noam M. Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Benton C. Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier García, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Oliveira Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathleen S. Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *ArXiv preprint*, abs/2204.02311, 2022. URL <https://arxiv.org/abs/2204.02311>.

## A Supplemental Material

### A.1 Fast convolution via FFT

For  $u, v \in \mathbb{C}^{1 \times L}$  the Circular Convolution Theorem states that,

$$\text{invFFT}_L(\text{FFT}_L(u) * \text{FFT}_L(v)) = v \cdot \begin{bmatrix} u_0 & u_1 & \cdots & u_{L-1} \\ u_{L-1} & u_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_1 \\ u_1 & \cdots & u_{L-1} & u_0 \end{bmatrix} = v \cdot \text{circulant}(u).$$

where  $*$  denotes elementwise multiplication. As FFT, invFFT can be done in  $O(L \log L)$  time this provides a fast algorithm for circulant matrix-vector product. In practice, linear systems can often be expressed as a circulant matrix-vector product and is also true in the case of Equation 4 which can be equivalently expressed as

$$[y_0 \dots y_{L-1} | 0 \dots 0]_{1 \times 2L} = [\bar{K} | 0 \dots 0]_{1 \times 2L} \cdot \text{circulant}([u_0 \dots u_{L-1} | 0 \dots 0])_{2L \times 2L}.$$

### A.2 Implementation of GSS

```

def simplified_dss_kernel(H, L, N=512):
    # Lambda_re, Lambda_im: [N]
    # C_re, C_im: [H N]
    Lambda = -Lambda_re.exp() + 1j*Lambda_im.exp()    # [N]
    C = C_re + 1j*C_im                                # [H N]
    S = (Lambda * arange(L).view(1,L)).exp()           # [N L]
    C = C * (Lambda.exp() - 1) / Lambda                # [H N]
    return einsum('hn,nl->hl', C, S).real            # [H L]

def dss(u, H, L):
    u = norm(u)
    # compute H state space kernels
    K = simplified_dss_kernel(H, L)
    K_f = rfft(K, pad_to=2*L)
    u_f = rfft(u, pad_to=2*L)
    y = irfft(K_f * u_f)[..., :L]
    # param D: [H, 1]
    return y + D * u

def gss(x, F=4096, L=4096, E=1024, H=256):
    shortcut, x = x, norm(x)
    v = dense(x, F, activation='gelu')
    u = dense(x, H, activation='gelu')
    y = dss(u, H, L)
    uc = dense(y, F)
    o = dense(uc * v, E)
    return o + shortcut

```

Figure 2: Pseudocode of GSS (§3.2).

# Atlas: Few-shot Learning with Retrieval Augmented Language Models

Gautier Izacard<sup>\*,♦,♣,♡</sup>

Patrick Lewis<sup>\*,♦</sup>

Maria Lomeli<sup>♦</sup>

Lucas Hosseini<sup>♦</sup>

Fabio Petroni<sup>♦</sup>

Timo Schick<sup>♦</sup>

Jane Dwivedi-Yu<sup>♦</sup>

Armand Joulin<sup>♦</sup>

Sebastian Riedel<sup>♦,♣</sup>

Edouard Grave<sup>♦</sup>

<sup>♦</sup> Meta AI Research, ♣ ENS, PSL University, ♡ Inria, ♠ University College London

gizacard@fb.com

plewis@fb.com

marialomeli@fb.com

hosss@fb.com

fabiopetroni@fb.com

schick@fb.com

janeyu@fb.com

ajoulin@fb.com

sriedel@fb.com

egrave@fb.com

## Abstract

Large language models have shown impressive few-shot results on a wide range of tasks. However, when knowledge is key for such results, as is the case for tasks such as question answering and fact checking, massive parameter counts to store knowledge seem to be needed. Retrieval augmented models are known to excel at knowledge intensive tasks without the need for as many parameters, but it is unclear whether they work in few-shot settings. In this work we present ATLAS, a carefully designed and pre-trained retrieval augmented language model able to learn knowledge intensive tasks with very few training examples. We perform evaluations on a wide range of tasks, including MMLU, KILT and NaturalQuestions, and study the impact of the content of the document index, showing that it can easily be updated. Notably, ATLAS reaches over 42% accuracy on Natural Questions using only 64 examples, outperforming a 540B parameters model by 3% despite having 50x fewer parameters.

## 1 Introduction

Large language models (LLMs) are impressive few-shot learners (Brown et al., 2020; Rae et al., 2021; Hoffmann et al., 2022; Chowdhery et al., 2022). They are able to learn new tasks with very few examples or even from instructions alone. For this generalisation ability to emerge, the key ingredients are scaling both the parameter count of the model, and the size of the training data. Large language models owe this improvement to both a larger computational budget, enabling more complex reasoning, and the ability to memorize more information related to downstream tasks from the larger training data. While it is intuitive to assume that increased reasoning abilities lead to better generalisation, and hence few-shot learning, the same is not true for in-parameter memorisation. Specifically, it is unclear to what extent effective few-shot learning requires vast knowledge in the parameters of the model.

In this paper, we investigate whether few-shot learning requires models to store a large amount of information in their parameters, and if memorisation can be decoupled from generalisation. To do so, we leverage the fact that memory can be outsourced and replaced by an external non-parametric knowledge source by employing a *retrieval-augmented* architecture. These models employ a non-parametric memory, e.g. a neural retriever over a large, external, potentially non-static knowledge source to enhance a parametric language model. In addition to their memorisation abilities, such architectures are attractive due to a number of other established advantages in terms of adaptability, interpretability and efficiency (Guu et al., 2020; Lewis et al., 2020; Yogatama et al., 2021; Borgeaud et al., 2021, *inter alia*). However, retrieval-augmented models have yet to

\*equal contribution

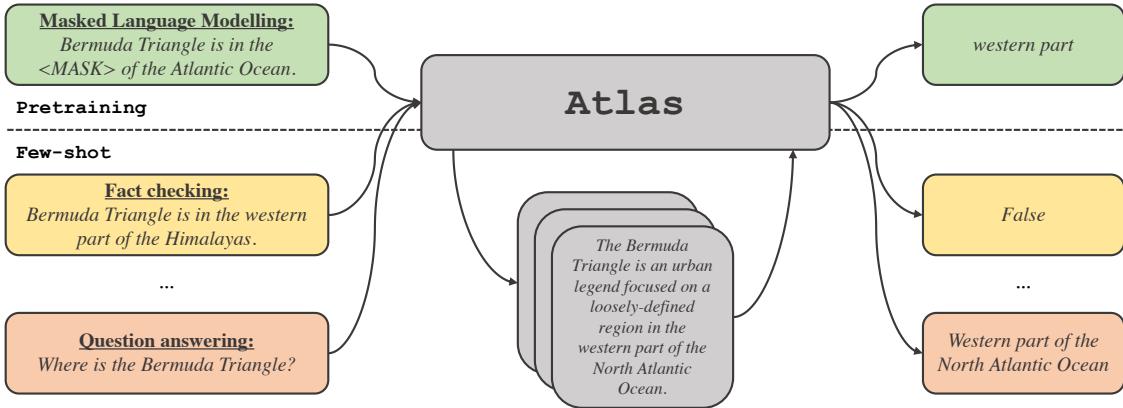


Figure 1: We introduce ATLAS, a retrieval-augmented language model that exhibits strong few-shot performance on knowledge tasks, and uses retrieval during both pre-training and fine-tuning.

demonstrate compelling few-shot learning capabilities. In this work we address this gap, and present ATLAS, a retrieval-augmented language model capable of strong few-shot learning, despite having lower parameter counts than other powerful recent few-shot learners.

ATLAS retrieves relevant documents based on the current context by using a general-purpose dense retriever using a dual-encoder architecture, based on the Contriever (Izacard et al., 2022). The retrieved documents are processed, along with the current context, by a sequence-to-sequence model using the Fusion-in-Decoder architecture (Izacard & Grave, 2020) that generates the corresponding output. We study the impact of different techniques to train ATLAS on its few-shot performance on a range of downstream tasks, including question answering and fact checking. We find that jointly pre-training the components is crucial for few-shot performance, and we carefully evaluate a number of existing and novel pre-training tasks and schemes for this purpose. ATLAS achieves strong downstream performance in both few-shot and resource-rich settings. For example, with only 11B parameters, ATLAS achieves an accuracy of 42.4% on NaturalQuestions using 64 training examples (45.1% with a Wikipedia-only index), outperforming PaLM (Chowdhery et al., 2022), a 540B parameter model by almost 3 points, and 64.0% in a full-dataset setting with a Wikipedia index, establishing a new state of the art by 8 points.

In summary we make the following contributions:

- A thorough study on how to design and train retrieval-augmented language models, with a focus on downstream few-shot learning and sample efficiency.
- The findings of this study lead to a retrieval-augmented language model, called ATLAS, that exhibits few-shot abilities that emerge at lower scale than standard LLM.
- We provide an exploration of fine-tuning strategies to efficiently adapt both the retriever and the language model to the task at hand.
- Thorough downstream experiments in few-shot settings, demonstrating state-of-the-art results on few-shot NaturalQuestions (+2.8%), TriviaQA (+3.3%), FEVER (+5.1%), and results on par or stronger than models with 15 $\times$  more parameters on MMLU.
- Experiments investigating full-dataset finetuning, setting new state-of-the-art results in NaturalQuestions (+8.1%), TriviaQA (+9.3%) and 5 KILT Tasks.
- Experiments demonstrating the updatability and interpretability characteristics of ATLAS.
- Experiments demonstrating that a compressed index using product quantisation achieves comparable performance as an uncompressed index while resulting in a 5x memory reduction.

Our code, pretrained ATLAS checkpoints, and various supporting data are available at <https://github.com/facebookresearch/atlas>

Task	Query	Output
Fact Checking	<i>Bermuda Triangle is in the western part of the Himalayas.</i>	<i>False</i>
Question Answering	<i>who is playing the halftime show at super bowl 2016</i>	<i>Coldplay</i>
Entity Linking	<i>NTFS-3G is an open source &lt;E&gt;cross-platform&lt;/E&gt; implementation of the Microsoft Windows NTFS file system with read-write support.</i>	<i>Cross-platform software</i>

Figure 2: Examples of query and output pairs for different tasks from KILT.

## 2 Method

Our approach follows the *text-to-text* framework (Raffel et al., 2019). This means that all the tasks are framed as follows: the system gets a *text query* as input, and generates a *text output*. For example, in the case of question answering, the query corresponds to the question and the model needs to generate the answer. In the case of classification tasks, the query corresponds to the textual input, and the model generates the lexicalized class label, i.e. the word corresponding to the label. We give more examples of downstream tasks, from the KILT benchmark in Figure 2. As many natural language processing tasks require *knowledge*, our goal is to enhance standard text-to-text models with retrieval, which, as we hypothesise in the introduction, may be crucial to endow models with few-shot capabilities.

### 2.1 Architecture

Our model is based on two sub-models: the *retriever* and the *language model*. When performing a task, from question answering to generating Wikipedia articles, our model starts by retrieving the top-k relevant documents from a *large corpus* of text with the retriever. Then, these documents are fed to the language model, along with the query, which in turns generates the output. Both the retriever and the language model are based on pre-trained transformer networks, which we describe in more detail below.

**Retriever.** Our retriever module is based on the Contriever (Izacard et al., 2022), an information retrieval technique based on continuous dense embeddings. The Contriever uses a dual-encoder architecture, where the query and documents are embedded independently by a transformer encoder (Huang et al., 2013; Karpukhin et al., 2020). Average pooling is applied over the outputs of the last layer to obtain one vector representation per query or document. A similarity score between the query and each document is then obtained by computing the dot product between their corresponding embeddings. The Contriever model is pre-trained using the MoCo contrastive loss (He et al., 2020), and uses unsupervised data only. As shown in the following section, an advantage of dense retrievers is that both query and document encoders can be trained without document annotation, using standard techniques such as gradient descent and distillation.

**Language model.** For the language model, we rely on the T5 sequence-to-sequence architecture (Raffel et al., 2019). We rely on the Fusion-in-Decoder modification of sequence-to-sequence models, and process each document independently in the encoder (Izacard & Grave, 2020). We then concatenate the outputs of the encoder corresponding to the different documents, and perform cross-attention over this single sequence in the decoder. Following Izacard & Grave (2020), we concatenate the query to each document in the encoder. Another way to process the retrieved documents in the language model would be to concatenate the query and all the documents, and to use this long sequence as input of the model. Unfortunately, this approach does not scale with the number of documents, since the self-attention in the encoder results in a quadratic complexity with respect to the number of documents.

## 2.2 Training objectives for the retriever

In this section, we discuss four different loss functions to train the retriever jointly with the language model. We consider loss functions that leverage the language model to provide *supervisory signal* to train the retriever. In other words, if the language model finds a document useful when generating the output, the retriever objective should encourage the retriever to rank said document higher. This allows us to train models using only query and output pairs from the task of interest, without relying on document annotations. For example, in the case of fact checking, a model only requires pairs of claims and corresponding verdicts but no documents containing the evidence to back up the verdict. In practice, we can apply this approach on any task, including self-supervised pre-training. As shown in the experimental section, pre-training is critical for obtaining models that exhibit few-shot learning abilities.

**Attention Distillation (ADist).** The first loss that we consider is based on the attention scores of the language model, and is heavily inspired by [Izacard & Grave \(2021\)](#). The main idea is that the cross-attention scores between the input documents and the output, can be used as a proxy of the importance of each input document when generating the output. In particular, [Izacard & Grave \(2021\)](#) showed that these scores can be aggregated across attention heads, layers and tokens for a given document to obtain a single score for each document. Then, these scores can be distilled into the retriever by minimizing the KL-divergence with the probability distribution  $p_{\text{RETR}}$  over the top-K documents  $\{\mathbf{d}_k\}_{1,\dots,K}$  obtained from the retriever:

$$p_{\text{RETR}}(\mathbf{d} \mid \mathbf{q}) = \frac{\exp(s(\mathbf{d}, \mathbf{q})/\theta)}{\sum_{k=1}^K \exp(s(\mathbf{d}_k, \mathbf{q})/\theta)}, \quad (1)$$

where  $s$  is the dot-product between the query and documents vectors and  $\theta$  is a temperature hyper-parameter.

In the original paper, it was proposed to use the pre-softmax scores from the decoder cross-attentions, and average across heads, layers and tokens. Here, we propose an alternative which gives slightly stronger results, which relies on the following observation. In the attention mechanism, as defined by

$$\mathbf{y} = \sum_{n=1}^N \alpha_n \mathbf{v}_n,$$

the contribution to the output  $\mathbf{y}$  of a particular token  $n$  cannot be evaluated from the attention score  $\alpha_n$  alone, but should also take the norm of the value  $\mathbf{v}_n$  into account. Hence, we use the quantity  $\alpha_n \|\mathbf{v}_n\|_2$  as the measure of relevance for token  $n$ . Following [Izacard & Grave \(2021\)](#), we average these scores over all attention heads, layers, and tokens to obtain a score for each document. We apply the SOFTMAX operator over the resulting scores, to obtain a distribution  $p_{\text{ATTN}}(\mathbf{d}_k)$  over the top-K retrieved documents. We then minimize the KL-divergence between  $p_{\text{ATTN}}(\mathbf{d}_k)$ , and the distribution  $p_{\text{RETR}}$  from the retriever defined in Equation 1:

$$\text{KL}(p_{\text{ATTN}} \parallel p_{\text{RETR}}) = \sum_{k=1}^K p_{\text{ATTN}}(\mathbf{d}_k) \log \left( \frac{p_{\text{ATTN}}(\mathbf{d}_k)}{p_{\text{RETR}}(\mathbf{d}_k)} \right).$$

Here, this loss is only used to optimize the parameters of the retriever, and not the language model. When using recent deep learning frameworks, this is achieved by applying a STOPGRADIENT operator on  $p_{\text{ATTN}}$ .

**End-to-end training of Multi-Document Reader and Retriever (EMDR<sup>2</sup>).** Next, we consider the method introduced by [Sachan et al. \(2021\)](#), which is inspired by the expectation-maximization algorithm, treating retrieved documents as latent variables. Given a query  $\mathbf{q}$ , the corresponding output  $\mathbf{a}$  and the set  $\mathcal{D}_K$  of top-K retrieved documents with the current retriever, the EMDR<sup>2</sup> loss to train the retriever is

$$\log \left[ \sum_{k=1}^K p_{\text{LM}}(\mathbf{a} \mid \mathbf{q}, \mathbf{d}_k) p_{\text{RETR}}(\mathbf{d}_k \mid \mathbf{q}) \right],$$

where  $p_{\text{RETR}}$  is again the probability over the top-K documents obtained with the retriever, as defined by Equation 1. Again, only the parameters of the retriever are updated by applying a STOPGRADIENT operator

---

around  $p_{LM}$ . One should note that the probability distribution over documents that maximizes this loss function is an indicator of the document corresponding to the highest probability of the output according to the language model. Finally, in practice, the EMDR<sup>2</sup> loss function is applied at the token level, and not at the sequence level.

**Perplexity Distillation (PDist).** Third, we discuss a simpler loss function which is loosely inspired by the objectives from the attention distillation and EMDR<sup>2</sup> methods ([Izacard & Grave, 2021](#); [Sachan et al., 2021](#)). More precisely, we want to train the retriever to predict how much each document would improve the language model perplexity of the output, given the query. To this end, we minimize the KL-divergence between the documents distribution of the retriever (Eqn. 1), and the documents posterior distribution according to the language model, using a uniform prior:

$$p_k \propto p_{LM}(\mathbf{a} \mid \mathbf{d}_k, \mathbf{q}).$$

Using the SOFTMAX operator, we have that

$$p_k = \frac{\exp(\log p_{LM}(\mathbf{a} \mid \mathbf{d}_k, \mathbf{q}))}{\sum_{i=1}^K \exp(\log p_{LM}(\mathbf{a} \mid \mathbf{d}_i, \mathbf{q}))}.$$

**Leave-one-out Perplexity Distillation (LOOP).** Finally, we propose an objective based on how much *worse* the prediction of the language model gets, when *removing* one of the top-k retrieved documents. To do so, we compute the log probability of the output for each subset of k-1 documents, and use the negative value as relevance score for each document. Following the previous loss function, we use the softmax operator to obtain a probability distribution over documents:

$$p_{\text{LOOP}}(\mathbf{d}_k) = \frac{\exp(-\log p_{LM}(\mathbf{a} \mid \mathcal{D}_K \setminus \{\mathbf{d}_k\}, \mathbf{q}))}{\sum_{i=1}^K \exp(-\log p_{LM}(\mathbf{a} \mid \mathcal{D}_K \setminus \{\mathbf{d}_i\}, \mathbf{q}))}.$$

As before, we then minimize the KL-divergence between this distribution, and the one obtained with retriever. This loss is more expensive to compute than PDist and EMDR, but, like ADist, employs the language model more closely to the way it is trained i.e. the LM is trained to be conditioned on a set of  $K$  documents. For LOOP, the language model is conditioned on  $(K - 1)$  documents, rather than a single document as in EMDR<sup>2</sup> and PDist.

For all losses, we can also use a temperature hyper-parameter when computing the target or retriever distributions to control the distribution’s peakiness of, which might be important for some tasks or losses. Indeed, for PDist and LOOP, the perplexity of the output may not vary much when conditioning on different documents, especially in the case of long outputs.

### 2.3 Pretext tasks

In this section, we describe pretext tasks that can be used to jointly pre-train the retriever and the language model using only unsupervised data.

**Prefix language modeling.** First, we consider a standard language modeling task as potential pre-training objective. To cast language modeling in the text-to-text framework, we consider a chunk of  $N$  words, and split this chunk in two sub-sequences of equal length  $N/2$ . Then, the first sub-sequence is used as the query, and the second corresponds to the output. We thus retrieve relevant documents by using the first sub-sequence of  $N/2$  tokens, to generate the output.

**Masked language modeling.** Second, we consider masked language modeling, as formulated by [Raffel et al. \(2019\)](#). Again, starting from a chunk of  $N$  words, we sample  $k$  spans of average length 3 tokens, leading to a masking ratio of 15%. We then replace each span by a different special token. The model is then trained to generate the masked spans, each span beginning with the special sentinel mask token that was inserted in the input sequence. We retrieve documents using the masked query, but replace the special mask tokens with a mask token supported by the retriever vocabulary.

---

**Title to section generation.** Finally, we consider a more abstractive generation task, generating sections from Wikipedia articles, given the article and section title. Here, the query corresponds to the title of the article, together with the title of the section, and the output corresponds to the text of the section. We exclude sections “See also”, “References”, “Further reading” and “External links”.

## 2.4 Efficient retriever fine-tuning

Retrieval is facilitated by using a document *index*, which is a pre-computed collection of the document embeddings for all the documents in the retrieval corpus. When jointly training the retriever and language model, the index needs to be updated regularly, otherwise, the embeddings of the documents stored in the index become stale relative to the updated retriever. This means that we need to recompute the embeddings for the full collection of documents regularly during training to keep the index fresh, which can be computationally expensive for large indices. This is particularly true at *fine-tuning* time, where the number of training examples could be small relative to the number of documents in the index. Training the retriever could thus add an important computational overhead compared to standard language model finetuning. In this section, we analyse strategies that might make this process more efficient, alleviating the need to re-compute the embeddings of all the documents too often.

**Full index update.** Let us start by analysing the overhead due to updating the index, compared to using a fixed retriever. To compare the computation time of different models, we will make the following assumption: the time required to perform a forward pass on a document with a model of  $P$  parameters is  $O(P)$ . While this computation model may seem naive, the main assumption is that document sizes are constant.<sup>1</sup> Since we split long documents into passages with similar number of words, and use padding when processing documents of different sizes, this assumption is reasonable in practice. Let  $K$  be the number of documents that are retrieved and processed by the language model,  $P_{\text{LM}}$  be the number of parameters of the language model and  $B$  the batch size. Each training step has a complexity of  $4 \times B \times K \times P_{\text{LM}}$ .<sup>2</sup>

Next, let  $N$  be the number of documents in the index, and  $P_{\text{RETR}}$  be the number of parameters of the retriever. Then, re-computing the full index has a complexity of  $N \times P_{\text{RETR}}$ . If we refresh the index every  $R$  training steps, we obtain the following overhead:

$$\frac{N \times P_{\text{RETR}}}{4 \times B \times K \times P_{\text{LM}} \times R}.$$

If we use the BERT-base architecture for our retriever and T5-XL for our language model, we get  $\frac{P_{\text{RETR}}}{P_{\text{LM}}} \approx \frac{1}{25}$ , leading to the overhead:

$$\frac{N}{100 \times B \times K \times R}.$$

If we use an index containing  $37M$  documents (the size of our Wikipedia index), train with a batch size of 64 with 20 retrieved documents and refresh the index every 1000 steps, this results in an overhead of  $\sim 30\%$ .

**Re-ranking.** A second strategy is to retrieve a larger number of documents  $L$  with the retriever, and to re-embed and rerank these documents with the up-to-date retriever, and pass the resulting top- $K$  to the language model. In that case, the overhead of reranking the top- $L$  documents is equal to  $B \times L \times P_{\text{RETR}}$ . Since we perform this operation at every time step, the overhead is equal to

$$\frac{L \times P_{\text{RETR}}}{4 \times K \times P_{\text{LM}}}.$$

Using the same assumption as before, we finally get that the overhead is of the order of  $\frac{L}{100 \times K}$ . If we re-rank 10x more documents than what the language model processes (i.e.,  $L = 10 \times K$ ), we get an overhead of 10%. However, note that if many updates are performed on the retriever, the index might still need to be fully

<sup>1</sup>See Hoffmann et al. (2022) for more details about the computation of the FLOPS corresponding to the forward and backward passes of transformer networks.

<sup>2</sup>There is a factor 4 to account for the backward pass and activation checkpointing.

---

updated, as the true top-k documents may not be retrieved in the top-L results from the stale index. In practice, it is possible to track the positions of the top-K re-ranked documents in the top-L, and estimate when the index needs to be updated.

**Query-side fine-tuning.** Finally, the last strategy is to decouple the encoding of the queries and documents. In this case, we fix the parameters corresponding to the document encoder, and only train the parameters corresponding to the query encoder. Thus, the embeddings of documents are fixed, and we do not need to refresh the index, and thus there is no computational overhead. As we will see in practice, the impact of fixing the documents encoder varies greatly for different tasks when a large training dataset is available. For most of the few-shot settings that we consider, query-side finetuning does not have large performance impact, and sometimes even slightly improves performance.

### 3 Related work

#### 3.1 Retrieval in natural language processing

**Retrieval for knowledge intensive tasks.** Previous work has shown that retrieval improves performance across a variety of tasks such as question answering (Voorhees et al., 1999; Chen et al., 2017; Kwiatkowski et al., 2019), fact checking (Thorne et al., 2018), dialogue (Dinan et al., 2019) or citation recommendation (Petroni et al., 2022). Historically, this information retrieval step was implemented using term-matching methods, such as TF-IDF or BM25 (Jones, 1972; Robertson et al., 1995). For open-domain question answering (Voorhees et al., 1999), documents are often retrieved from Wikipedia (Chen et al., 2017). Recently, dense retrievers based on neural networks have become popular. These usually follow a dual-encoder architecture (Yih et al., 2011; Huang et al., 2013; Shen et al., 2014), where queries and passages are encoded independently as vectors, and relevance is computed using the inner product or Euclidean distance. Popular supervised retrievers include DPR (Karpukhin et al., 2020), which is trained to discriminate the relevant passage among negative passages, and extensions such as ANCE (Xiong et al., 2020) which improved the hard negatives mining process. We refer the reader to Yates et al. (2021) for a survey of dense retrieval techniques.

After retrieval, the relevant documents are processed to produce the final output. In open-domain QA, models can extract a span of text from retrieved documents as the answer (Chen et al., 2017; Clark & Gardner, 2018; Wang et al., 2019; Karpukhin et al., 2020), a method inspired by reading comprehension (Richardson, 2013; Rajpurkar et al., 2016). Recently, generating the answer as free-form text, using a seq2seq model conditioned on retrieved documents have become prevalent (Lewis et al., 2020; Izacard & Grave, 2020; Min et al., 2020). These architectures have also been shown to reduce hallucination in dialogue agents (Shuster et al., 2021).

**Retriever training.** The need for expensive query-document annotations for training the retriever can be bypassed, by leveraging signals from the language model, or using unsupervised learning. REALM (Guu et al., 2020) and RAG (Lewis et al., 2020) jointly train the retriever and language model by modelling documents as latent variable, and minimizing the objective with gradient descent. REALM pre-trains end-to-end with an MLM approach but uses an extractive BERT-style model (Devlin et al., 2019). Guu et al. (2020) also explore a query-side finetuning at finetuning time to avoid index refreshes, which is also explored in the context of phrase-based retrieval by Lee et al. (2021b). Izacard & Grave (2020) proposed to use cross-attention scores as supervision with knowledge distillation. Sachan et al. (2021) perform joint training of the reader and the retriever by leveraging the perplexity of the output generated by the reader. Sachan et al. (2021) and Lee et al. (2021a) both employ salient span masking to pre-train retrievers, leveraging the perplexity and attention scores from the language model. The *inverse cloze task* was proposed by Lee et al. (2019) to pre-train dense retrievers in an unsupervised way. Paranjape et al. (2021) propose a method to train retrieval-augmented generators using a second “informed” retriever with access to the output, which the test-time retriever can be distilled from, and Hofstätter et al. (2022) recently proposed a training set filtering/weighting approach to train stronger retrieval-augmented generators. Izacard et al. (2022) explored different contrastive learning methods to train retrievers, while Ram et al. (2022) used recurring spans within a document to create pseudo-positive query-document pairs.

---

**Retrieval-augmented language models.** Continuous cache models (Grave et al., 2017b) defines a probability distribution over recent tokens, by computing the similarity between previous and current representations of tokens. This distribution is then interpolated with the distribution of the language model, to improve predictions. Later, the amount of tokens used to compute this distribution was extended to a much larger memory by leveraging approximate nearest neighbors search (Grave et al., 2017a). The related kNN-LM model (Khandelwal et al., 2020) replaced LSTMs by transformer networks, and scaled the memory to billions of tokens, leading to strong performance improvements. More recently, RETRO (Borgeaud et al., 2021) extended these by scaling the retrieval memory to trillions of tokens, and changing the model architecture to take retrieved documents as input.

**Retrieval-Augmentation with Search Engines.** Recently, different works have proposed to train large language models to interact with a search engine, by generating text queries, and using the retrieved documents as additional context (Nakano et al., 2021; Thoppilan et al., 2022; Shuster et al., 2022). In the context of few-shot question answering, Lazaridou et al. (2022) used the question to perform a search query, and retrieved documents are added to the prompt of a large language model performing in-context learning.

### 3.2 Few-shot learning

Few-shot learning, the task of learning from very few examples, has been studied for decades (Thrun & Pratt, 1998; Fink, 2005; Vinyals et al., 2016), but has recently seen an explosion of interest in NLP with the arrival of large pre-trained models, which exhibit emergent few-shot learning abilities (Wei et al., 2022).

**In-context Learning with large Language models.** Providing language models with natural language descriptions of tasks, as proposed by Radford et al. (2019) has led to significant developments in few-shot learning. GPT-3 (Brown et al., 2020) demonstrated the ability of large language models to perform few-shot predictions, where the model is given a description of the task in natural language with few examples. Scaling model size, data and compute is crucial to enable this learning ability, leading to the further development of large models (Lieber et al., 2021; Rae et al., 2021; Smith et al., 2022; Chowdhery et al., 2022; Smith et al., 2022). Hoffmann et al. (2022) revisited the scaling law from Kaplan et al. (2020), suggesting that training on more data with a smaller model may be more effective, resulting in Chinchilla, a 70B parameter model with improved parameter efficiency.

**Few-shot finetuning and prompt-based learning.** The above models perform few-shot learning with in-context instructions without training the parameters of the language model. Few-shot learning can also be accomplished by combining textual templates (“prompts”) and various forms of model finetuning, either fully updating a model’s parameters, e.g. for classification (Schick & Schütze, 2021a; Schick & Schutze, 2021; Gao et al., 2021; Tam et al., 2021) or generation (Schick & Schütze, 2021b). Prompts themselves can be optimized, for example by search (Jiang et al., 2020; Shin et al., 2020) or by only updating parts of the model (Logan et al., 2021), or learning “soft-prompts” (Lester et al., 2021; Li & Liang, 2021). Due to its simplicity, in this work we either employ simple prompts or simply feed in inputs without preprocessing, and perform full-model finetuning, a method similar to Le Scao & Rush (2021).

## 4 Experiments

In this section, we report empirical evaluations of our language models on few-shot learning. We start by introducing our experimental setup, describing our evaluation benchmarks in section 4.1, and giving the training details of our models in section 4.2. Then, we perform an ablation study to compare the different technical choices leading to our main model. We finally evaluate this model, called ATLAS, on different natural language understanding tasks in few-shot and full dataset settings.

### 4.1 Benchmarks

To evaluate our retrieval-augmented language models we consider the following benchmarks, which include different tasks.

---

**Knowledge-Intensive Language Tasks (KILT).** First, we use the KILT evaluation suite (Petroni et al., 2020), containing 11 datasets corresponding to 5 tasks: fact checking, question answering, dialog generation, entity linking and slot-filling. These different tasks require knowledge about the world to be solved, which can be found in Wikipedia. We evaluate our model on the following tasks and datasets included in KILT: question answering: NaturalQuestions (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017) and HotpotQA (Yang et al., 2018); slot filling: Zero Shot RE (Levy et al., 2017) and T-REx (Elsahar et al., 2018); entity linking: AIDA CoNLL-YAGO (Hoffart et al., 2011); dialogue: Wizard of Wikipedia (Dinan et al., 2019); and fact checking: FEVER (Thorne et al., 2018). The KILT versions of these datasets differ from their original versions, as instances requiring knowledge not present in the August 2019 Wikipedia dump have been removed.

**Massively-Multitask Language Understanding (MMLU).** Our second main evaluation benchmark is MMLU (Hendrycks et al., 2021), which contains 57 multi-choice question answering datasets (referred to as domains), sourced from real examinations designed for humans. These cover a very broad range of topics, e.g. high school mathematics, professional law, logical fallacies and clinical knowledge and can be broadly categorized in four subsets: humanities, social sciences, STEM and “other”. We focus on few-shot learning, and the authors of the benchmarks suggest to use 5 training examples per domain. Beyond the 5-shot setting, We also consider three additional settings. The first is a *zero-shot* setting, with no training data at all. The second, which we call *multi-task few-shot*, is where we train a single model on the 6-shot data from all tasks, hence leading to a training set of 285 examples. The last, which we call *transfer learning*, leverages additional training examples from other multiple-choice QA tasks provided by the MMLU authors, namely MCTest (Richardson, 2013), RACE (Lai et al., 2017), ARC (Clark et al., 2018) and OBQA (Mihaylov et al., 2018) leading to a training set of 95k examples.

**Additional benchmarks.** Additionally, we report results on the original open-domain versions of the popular NaturalQuestions (Kwiatkowski et al., 2019), and TriviaQA (Joshi et al., 2017) datasets. We also evaluate our model on the original version of FEVER (Thorne et al., 2018), which presents fact checking as a three-way classification problem for textual claims (either “Supported”: the text is supported by evidence in Wikipedia, “refuted”: the claim is not consistent with evidence in Wikipedia, or “not enough info”, where there is insufficient evidence to make a judgement). We also perform experiments to assess temporal sensitivity of our models. Here, we construct a dataset from TempLAMA (Dhingra et al., 2022), consisting of a set of time-sensitive cloze questions on a range of topics, where the answer changes from 2017 to 2020. We assess the accuracy of our models when supplied with a index from 2017 vs 2020 to assess to what degree models faithfully reflect the content of the index supplied to them at test time, and how effective updating the index is as a *continual learning* or model updateability method.

## 4.2 Technical details

We now describe the procedure for pre-training and fine-tuning our models. We focus on the setting used for the ablation studies performed in Section 4.3 and Section 4.4. We give more details about the hyperparameters used for our final model later.

**Pre-training.** For the pre-training, we initialize the retriever module using the unsupervised *Contriever* model, which uses the BERT-base architecture. We initialize the language model with the T5 pre-trained weight. As the original T5 pre-trained model included supervised data in the training set, we use the version 1.1 models which were trained on unlabeled text only. Specifically, we initialize from the **T5-1m-adapt** variants due to their improved stability.

For the ablation studies performed in Section 4.3 and Section 4.4, we use T5-XL which contains 3B weights. We pre-train all our models for 10,000 iterations, using AdamW with a batch size of 64 and a learning rate of  $10^{-4}$  for the reader and  $10^{-5}$  for the retriever with linear decay and 1,000 warmup steps. We refresh the index every 1,000 steps. This means that the index is recomputed 10 times during the pre-training, leading to an overhead of around 30%, compared to training with a fixed retriever. We set the number of retrieved documents to 20. We detail the hyperparameters used for the training of our final model at the beginning of Section 4.5.

Table 1: **Retriever loss ablation.** We compare different loss functions to pre-train the retriever jointly with the language model. We use the prefix MLM task, and the December 2021 Wikipedia dump for both the index and pre-training data. Fine-tuning is performed with query-side fine-tuning and the loss used for pre-training. Best result is bold, second highest underlined.

	MLM	64-shot				1024-shot			
		NQ	WoW	FEVER	Avg.	NQ	WoW	FEVER	Avg.
Closed-book	1.083	6.5	14.1	59.0	26.5	10.7	16.5	75.3	34.2
No Joint pre-training	-	9.0	14.1	67.0	30.0	9.9	16.6	78.3	34.9
Fixed retriever	0.823	39.9	14.3	72.4	42.2	45.3	<u>17.9</u>	90.0	<u>51.1</u>
ADist	<u>0.780</u>	40.9	14.4	73.8	43.0	<u>46.2</u>	17.2	<b>90.9</b>	<b>51.4</b>
EMDR <sup>2</sup>	0.783	<u>43.3</u>	<u>14.6</u>	72.1	43.3	44.9	<b>18.3</b>	85.7	49.6
PDist	0.783	<b>45.0</b>	<b>15.0</b>	<b>77.0</b>	<b>45.7</b>	44.9	<u>17.9</u>	<u>90.2</u>	51.0
LOOP	<b>0.766</b>	41.8	<b>15.0</b>	<u>74.4</u>	<u>43.7</u>	<b>47.1</b>	<u>17.9</u>	87.5	50.8

**Fine-tuning.** When performing a downstream task, either in a few-shot setting or with a large training set, we employ fine-tuning to adapt our models to these tasks. For the few-shot KILT ablation experiments, we perform a fixed number of fine-tuning iterations, instead of using early-stopping. More precisely, we decided to use 50 iterations for the 64-shot setting and 200 iterations in the 1024-shot setting. In both cases, we use a batch size of 32 examples, a learning rate of  $4 \times 10^{-5}$  with linear decay and 5 warmup steps for both the reader and the retriever.

**Unlabeled datasets.** Finally, we discuss the unlabeled text datasets that we use to train our models, which form the retrieval index. First, we consider the Dec. 20, 2021 Wikipedia dump, for which we keep the lists and infoboxes, which are linearized by adding a semi-colon separator between the entries. We split articles by section, and split long sections into passages of equal sizes and containing less than 200 words. This leads to a total of 37M passages, containing 78 words in average. We also use documents from the 2020-10 common crawl dump, preprocessed with the CCNet pipeline (Wenzek et al., 2020). We perform additional document filtering, in a similar fashion to Gopher (Rae et al., 2021). More precisely, we filter documents based on document length, average word length, ratio of alphanumeric characters and number of repeated tokens. This leads to a total of 350M passages. The same passages are used for the index and model pre-training. During pre-training, we ensure the passage we are training on is filtered out from the retrieved documents, to prevent the model from simply retrieving the passage it is de-nosing/generating, and trivially using it to solve the pre-training task.

### 4.3 Pre-training loss and tasks

We start our ablation study by comparing different pre-training tasks, and objective functions to jointly train the retriever and the language model. Our goal here is to answer the following research questions:

- (RQ 1) Does jointly pre-training the whole model lead to better few-shot performance?
- (RQ 2) What is the best objective function for the retriever, and the best pretext task?

We start by comparing the training objectives of the retriever, introduced in Section 2.2, by pre-training models using the masked language modelling task. We evaluate these models on a subset of the 64-shot and 1024-shot KILT benchmark: NaturalQuestions, FEVER and Wizard of Wikipedia, along with two baselines: a ‘closed-book’ (i.e. non-augmented T5) baseline, pre-trained on the same data, and initialized from `Contriever` and `T5-1m-adapt`. We report results in Table 1. First, we note the poor performance of the closed-book baseline, indicating the importance of augmentation. Next, we observe that pre-training our model with retrieval is important to obtain good performance on few-shot tasks. Indeed, all models that include retrieval during pre-training strongly outperform the baseline without joint pre-training. Next,

Table 2: **Pretext task ablation.** We compare different pretext tasks, used to jointly pre-train our models. Examples are randomly sampled from the training set of the KILT version of the dataset. We report the exact match on NaturalQuestions, the F1 score on Wizard of Wikipedia and the accuracy on FEVER.

	64-shot				1024-shot			
	NQ	WoW	FEVER	Avg.	NQ	WoW	FEVER	Avg.
Prefix Language Modelling	41.0	14.5	64.9	40.1	<b>44.7</b>	17.9	86.0	49.5
Masked Language Modelling	<b>42.7</b>	<b>14.9</b>	<b>69.7</b>	<b>42.4</b>	<b>44.7</b>	<b>18.3</b>	<b>88.8</b>	<b>50.6</b>
Title-to-section generation	41.1	15.2	66.1	40.8	45.4	17.9	84.6	49.3

Table 3: **Index content ablation.** In this table, we report results for models where the content of the index was changed between the pre-training and the fine-tuning.

Index	Training data	64-shot				1024-shot			
		NQ	WoW	FEVER	Avg.	NQ	WoW	FEVER	Avg.
Wiki	Wiki	<b>42.7</b>	14.9	69.7	<b>42.4</b>	44.7	18.3	88.8	<b>50.6</b>
Wiki	CC	40.9	<b>15.3</b>	67.3	41.2	<b>44.8</b>	<b>18.4</b>	88.1	50.4
CC	Wiki	32.9	14.5	<b>72.1</b>	39.8	37.8	17.1	85.8	46.9
CC	CC	38.4	14.9	70.1	41.1	42.0	17.3	<b>88.9</b>	49.4

we compare a model that was pre-trained with a fixed retriever, and models using the various retriever training objectives. On the MLM validation metric corresponding to the pre-training objective, we observe that jointly training the retriever leads to strong improvements. This effect tends to be less marked on 64-shot downstream tasks, and almost non-existent for 1024-shot. We believe that this is evidence that the biggest impact of pre-training is on the language model, which learns to use and aggregate information from the retrieved documents. Lastly, we do not observe significant systematic differences between the different retriever training objectives. We thus decide adopt use Perplexity Distillation for subsequent experiments, as it tends to be more stable than EMDR<sup>2</sup> or ADist, and more computationally efficient than LOOP.

Next, we compare the different self-supervised pretext tasks introduced in Section 2.3 in Table 2. Here we observe similar results for all three tasks, with a small advantage for masked language modelling. Thus, in what follows, we adopt masked language modelling for pre-training.

Finally, we consider different combinations of data sources—Wikipedia and common crawl—for the index and training data during pre-training. In all cases, we use the Wikipedia 2021 dump as the index when performing few-shot fine-tuning. We report results in Table 3. First, we observe that using a Wikipedia-based index leads to better downstream performance. There could be two explanations for this: first, as we use Wikipedia for the few-shot tasks, the model might be better adapted when trained using the same data. Another explanation might be that Wikipedia is a higher-quality and denser source of knowledge than common crawl. Second, when using a common crawl index, we observe that pre-training on Wikipedia data leads to lower performance than using common crawl data. We believe that the primary reason is that the distribution mismatch between the two domains leads to generally-less relevant retrieved documents. In turn, this probably means that the pre-training is less efficient, because the language model does not leverage as much information from the documents. In the following, we thus decide to combine the data from both domains for both the index and the pre-training data.

#### 4.4 Fine-tuning

In this section, we perform an ablation study on how to apply our models on downstream tasks, which relies on fine-tuning. In particular, we want to investigate the following research question:

- (RQ 3) How to efficiently fine-tune ATLAS on tasks with limited training data?

Table 4: **Retriever fine-tuning ablation.** Here, we compare different strategies to fine-tune the retriever in a few-shot setting.

	64-shot				1024-shot			
	NQ	WoW	FEVER	Avg.	NQ	WoW	FEVER	Avg.
Standard fine-tuning	44.3	14.9	73.2	44.1	47.0	18.4	89.7	<b>51.7</b>
Top-100 re-ranking	44.2	14.6	75.4	<b>44.7</b>	<b>47.1</b>	<b>18.7</b>	88.9	51.6
Query-side fine-tuning	<b>45.0</b>	<b>15.0</b>	<b>77.0</b>	<b>45.7</b>	44.9	17.9	<b>90.2</b>	51.0
Fixed retriever	36.8	14.5	72.0	41.1	38.0	17.7	89.3	48.3

Table 5: **Performance on MMLU as a function of model size.**

	5-shot			5-shot (multi-task)			Full / Transfer		
	770M	3B	11B	770M	3B	11B	770M	3B	11B
Closed-book T5	29.2	35.7	36.1	26.5	40.0	43.5	42.4	50.4	54.0
ATLAS	38.9	42.3	43.4	42.1	48.7	56.4	56.3	59.9	65.8
$\Delta$	+9.8	+6.6	+7.3	+15.6	+8.7	+12.9	+13.9	+9.5	+11.8

To answer this question, we compare the different strategies to fine-tune the retriever module, described in Section 2.4. We report results in Table 4. First, as for pre-training, we observe that keeping the retriever fixed during fine-tuning leads to a significant performance drops, for both 64- and 1024-shot settings. Second, the re-ranking strategy (row 2) leads to very similar results to fully updating the index (row 1), while being significantly more efficient. Lastly, fine-tuning only the query encoder also leads to strong results: in particular, in the 64-shot setup, this is slightly stronger than performing full fine-tuning, which we attribute to there being less opportunity for over-fitting. On the other hand, on the 1024-shot setting, performing a full fine-tuning leads to stronger results, especially on NaturalQuestions. In the following, we thus use query-side fine-tuning for experiments with small numbers of examples, and standard fine-tuning for larger datasets.

#### 4.5 Training and evaluating Atlas

In this section, we apply the findings from the ablations of the previous sections to train a family of ATLAS models, ranging from 770M to 11B parameters. More specifically, we use the Perplexity Distillation objective function, along with the masked language modelling pretext task. We pre-train these models using a mix of Wikipedia and Common Crawl data, for both the training data and content of the index. We retrieve 20 documents, and update the index every 2,500 steps and perform re-ranking of the top-100 documents. We pre-train models for 10,000 iterations using AdamW with a batch size of 128.

##### 4.5.1 MMLU Results

As mentioned in section 4.1, we consider four setting for MMLU: 1) a zero-shot setting where we directly apply the pretrained model with no few-shot finetuning 2) a 5-shot setting, where we finetune a model using 5 training examples for each of the 57 domains 3) a 5-shot multitask setting, where, rather than finetuning a model independently for each domain, we train a single model to perform all tasks and 4) a setting with access to a number of auxiliary datasets, with 95K total training examples. We train the models to generate the letter corresponding to the correct answer option ('A', 'B', 'C' or 'D'), and pick the answer with the most likely of the 4 letters at test time. Full technical details can be found in appendix A.1.

**Performance vs Parameters.** We start by comparing ATLAS to closed-book models of different sizes for 5-shot, 5-shot multitask and the full setting, and report results in Table 5. Across these settings, ATLAS outperforms the closed-book baselines by between 6.6 and 15.6 points, demonstrating consistent utility of retrieval for few-shot language understanding across 57 domains. The closed-book T5 struggles to perform

---

Table 6: **Standard vs de-biased inference for MMLU** These results are reported for ATLAS-11B, using cyclic permutations for de-biasing, which increases inference costs by a factor of  $4\times$ .

	Zero-shot	5-shot	5-shot (multi-task)	Full / Transfer
Standard Inference	36.8	43.4	56.4	65.8
De-biased Inference	47.1	47.9	56.6	66.0

significantly better than random (25%) in few-shot settings with 770M parameters, whereas the equivalent ATLAS achieves around 40%, significantly better than random, despite its small size. All models improve with more data, but interestingly, the 770M models do not benefit as much from few-shot multitask learning compared to larger models (for closed-book, it actually loses 3 points) suggesting smaller models struggle to grasp the synergies between the tasks in the few-shot setting. Larger models exploit the multi-task setting well, with ATLAS improving more than closed-book. For example, ATLAS-11B improves by 13 points ( $43.4 \rightarrow 56.4$ ), but equivalent closed-book only improves by 7 ( $36.1 \rightarrow 43.5$ ). Finally, on the transfer learning setting, all models improve, but the relative gaps between closed-book at ATLAS models remain similar.

**De-biasing.** When finetuning, we permute which answer option appears with which answer letter to reduce over-fitting and encourage a uniform prior over answer letters. However, the model may still exhibit a bias towards some letters, especially in few-shot settings, so we also include a second ‘de-biased’ inference mode in addition to the standard inference used above. Here, we run 4 forward passes, one for each cyclic permutation of the answer letter-answer option assignment in the question, e.g. the answer option assigned to letter ‘A’ becomes ‘B’, what was ‘B’ becomes ‘C’ etc.<sup>3</sup> We then sum the 4 probabilities to obtain the final prediction, which reduces spurious bias towards one of the answer letters (further details in appendix A.1). The results are shown in Table 6. We find that in zero-shot and 5-shot settings, de-biasing is very effective, improving results by 10.3 and 4.5 points respectively. When more training data is available, the need for de-biasing decreases, leading to only 0.2 point improvement in the multi-task and full data settings.

**Comparison to published works** Next, we compare our ATLAS-11B results with de-biasing to recently reported results with state-of-the-art large language models such as GPT-3 or Chinchilla, which required significantly more amount of computation to train. We report results in Table 7. We find that ATLAS is able to perform significantly better than random in zero-shot, and in conjunction with de-biased inference, achieves zero-shot scores that exceed 5-shot results reported with GPT3 in the literature (47.1% vs 43.9%) (Hendrycks et al., 2021). For the 5-shot setting, ATLAS outperforms GPT-3 by 4%, while using  $15\times$  less parameters, and  $10\times$  less pre-training compute.<sup>4</sup> When multitask-training on the combined 5-shot data, ATLAS improves to 56.6% close to the 5-shot performance of Gopher (60.0%). Finally, on the full data setting, where we train on auxiliary data recommended by the MMLU authors, ATLAS reaches an overall accuracy of 65.6%, close to the state-of-the-art. Interestingly, in this setup, ATLAS significantly outperforms GPT-3, while on the 5-shot setting, their performance is similar.

#### 4.5.2 Open-domain Question Answering Results

Next we evaluate ATLAS on two open-domain question answering benchmarks: NaturalQuestions and TriviaQA. We compare to prior work, both in a few-shot setting using 64 examples, and using the full training set, and report results in Table 8. On these benchmarks, which require high-degree of memorisation, we clearly see the benefits of retrieval-augmentation. ATLAS-11B obtains state-of-the-art results on 64-shot question answering, for both NaturalQuestions and TriviaQA. In particular, it outperforms significantly larger models, such as PaLM, or models that required significantly more training compute such as Chinchilla. When using the full training set, ATLAS also obtains state-of-the-art results, for example improving the accuracy on NaturalQuestions from 55.9% to 60.4%. This result is obtained using an index comprised of CCNet and

<sup>3</sup>Exploring all answer option permutations would involve 24 forward passes, which improves results by an additional  $\sim 1\%$  over the 4 cyclic permutations, but requires much more compute, so we exclude it here, see Appendix A.1

<sup>4</sup>ATLAS’s pre-training compute is dominated by the T5 pre-training. The computational requirements for the retrieval-augmented pre-train is orders of magnitude lower

Table 7: **Comparison to state-of-the-art on MMLU.** \*For the 5-shot setting, ATLAS uses fine-tuning, while previous works use in-context learning. The ATLAS model uses de-biased inference. Train FLOPS refers to total the amount of computation necessary to train the model, including pre-training and/or fine-tuning.

Setting	Model	Params	Train FLOPS	All	Hum.	Soc. Sci.	STEM	Other
zero-shot	ATLAS	11B	3.5e22	47.1	43.6	54.1	38.0	54.4
5-shot	GPT-3	175B	3.1e23	43.9	40.8	50.4	36.7	48.8
	Gopher	280B	5.0e23	60.0	56.2	71.9	47.4	66.1
	Chinchilla	70B	5.0e23	<b>67.5</b>	<b>63.6</b>	<b>79.3</b>	<b>55.0</b>	73.9
	ATLAS*	11B	3.5e22	47.9	46.1	54.6	38.8	52.8
5-shot (multi-task)	ATLAS	11B	3.5e22	56.6	50.1	66.4	46.4	66.2
Full / Transfer	UnifiedQA	11B	3.3e22	48.9	45.6	56.6	40.2	54.6
	GPT-3	175B	3.1e23	53.9	52.5	63.9	41.4	57.9
	ATLAS	11B	3.5e22	66.0	61.1	77.2	53.2	<b>74.4</b>

Table 8: **Comparison to state-of-the-art on question answering.** We report results on NaturalQuestions, and on TriviaQA for both the filtered set, commonly used for open-domain question answering and the unfiltered hidden set for which evaluation is accessible online: <https://competitions.codalab.org/competitions/17208>. For the 64-shot setting, our model uses fine-tuning, while the other models use prompting.

Model	NQ		TriviaQA filtered		TriviaQA unfiltered	
	64-shot	Full	64-shot	Full	64-shot	Full
GPT-3 (Brown et al., 2020)	29.9	-	-	-	71.2	-
Gopher (Rae et al., 2021)	28.2	-	57.2	-	61.3	-
Chinchilla (Hoffmann et al., 2022)	35.5	-	64.6	-	72.3	-
PaLM (Chowdhery et al., 2022)	39.6	-	-	-	81.4	-
RETRO (Borgeaud et al., 2021)	-	45.5	-	-	-	-
FiD (Izacard & Grave, 2020)	-	51.4	-	67.6	-	80.1
FiD-KD (Izacard & Grave, 2021)	-	54.7	-	73.3	-	-
R2-D2 (Fajcik et al., 2021)	-	55.9	-	69.9	-	-
ATLAS	<b>42.4</b>	<b>60.4</b>	<b>74.5</b>	<b>79.8</b>	<b>84.7</b>	<b>89.4</b>

the December 2021 Wikipedia corpora, our default setting for the index. In section 5.2 we consider using indexes composed of Wikipedia corpus archived at different dates, and demonstrate an additional +3.6% on NaturalQuestions when using an index which is temporally matched to NaturalQuestions. We report performance as a function of model size as well as detailed hyperparameters in Appendix A.2.

ATLAS also compares favorably to recent work exploring retrieval-augmented few-shot question answering with very large models. Lazaridou et al. (2022) explore NaturalQuestions in a 15-shot setup using Gopher, augmenting questions with 50 passages retrieved using Google Search. This method consists of generating 4 candidate answers from each retrieved passage, and then re-ranking using either a score inspired by RAG (Lewis et al., 2020) or a more expensive approach. This method (not shown in our tables) achieves exact match scores of 32.7% (RAG) and 38.4% (Ensemble), requiring 50 (RAG) or 450 (Ensemble) forward passes of Gopher-280B per test-time question. ATLAS, using the same 15 training examples and 50 passages achieves 38.7 EM, despite having 25× fewer parameters, and requiring comparatively negligible compute.

#### 4.5.3 FEVER Results

We report results on the original 3-class FEVER fact checking test set in Table 9. We consider a 64-shot setting, with training examples uniformly sampled from the full training set. Unlike the development and test sets, the train set is imbalanced, with more positive labels than negative, posing a challenge for few-shot

Table 9: **Comparison to state-of-the-art on FEVER.** We report accuracy on FEVER test set, for which evaluation is available here: <https://competitions.codalab.org/competitions/18814>. For the few-shot settings, our model uses fine-tuning while other models use prompting. <sup>†</sup>uses an index composed of the FEVER Wikipedia corpus.

	15-shot	65-shot	Full dataset
Gopher (Rae et al., 2021)	51.1	-	-
ProoFVer (Krishna et al., 2021)	-	-	79.5
ATLAS	<b>56.2</b>	<b>64.3</b>	78.0 / 80.1 <sup>†</sup>

Table 10: **Downstream results on the KILT hidden test sets** Downstream metrics are accuracy (AIDA, CONLL-YAGO, FEVER, T-REx, zero-shot RE), exact match (Natural Questions, HotpotQA, TriviaQA), or F1 (Wizard of Wikipedia).

Model	AIDA	FEV	T-REx	zsRE	NQ	HoPo	TQA	WoW
	ACC	ACC	ACC	ACC	EM	EM	EM	F1
GENRE (Cao et al., 2021)	89.9	-	-	-	-	-	-	-
Sphere (Piktus et al., 2021)	-	89.0	81.7	74.2	51.6	38.3	72.7	15.5
SEAL (Bevilacqua et al., 2022)	-	89.5	83.6	74.6	53.7	40.5	70.9	18.3
Re2G (Glass et al., 2022)	-	89.6	<b>87.7</b>	-	51.7	-	76.3	18.9
FID with RS (Hofstätter et al., 2022)	-	92.2	85.2	<b>83.7</b>	61.2	39.1	<b>84.6</b>	20.6
ATLAS, 64-shot	66.5	87.1	58.9	74.9	43.6	34.7	76.4	15.5
ATLAS, full train set	<b>90.6</b>	<b>93.5</b>	85.1	80.8	<b>61.3</b>	<b>50.6</b>	84.0	<b>21.6</b>

learning. In this setting, we achieve an accuracy of 64.3%. We also report a 15-shot setting, with 5 examples uniformly sampled from each class to compare with published results from Gopher (Rae et al., 2021), where ATLAS scores 56.2%, outperforming Gopher by 5.1 points. Lastly we fine-tune our model on the full training set, and achieve a score of 78%, within 1.5% of the ProoFVer, which uses a specialized architecture, a retriever trained with sentence-level annotations, and is supplied with the Wikipedia corpus released with FEVER, whereas ATLAS retrieves from CCNet and the December 2021 Wikipedia dump. If we give ATLAS an index comprised of the FEVER Wikipedia corpus, we set a new state-of-the-art of 80.1%

#### 4.5.4 KILT Results

Finally we evaluate ATLAS on KILT, a benchmark composed of several different knowledge intensive tasks, which was described in section 4.1. We report results on test sets in Table 10 for which evaluation is available online<sup>5</sup>. The KILT versions of datasets are filtered, and thus results for datasets we have evaluated elsewhere are not directly comparable on KILT (i.e. FEVER, NQ and TQA). We consider both a 64-shot setting and a full fine-tuning setting, in both cases we train ATLAS individually on each dataset. More details on the hyperparameters and development set results are reported in Appendix A.3. For 64-shot, we greatly exceed random performance, and are even competitive with some fully-finetuned models on the leaderboard, such as for FEVER, where our 64-shot ATLAS is only 2-2.5 points behind Sphere, SEAL and Re2G, and outperforms Sphere and SEAL on zero-shot RE. In the full dataset setting, ATLAS is within 3% to the state-of-the-art for 3 datasets, and sets the state-of-the-art in the remaining five datasets.

<sup>5</sup><https://eval.ai/web/challenges/challenge-page/689>

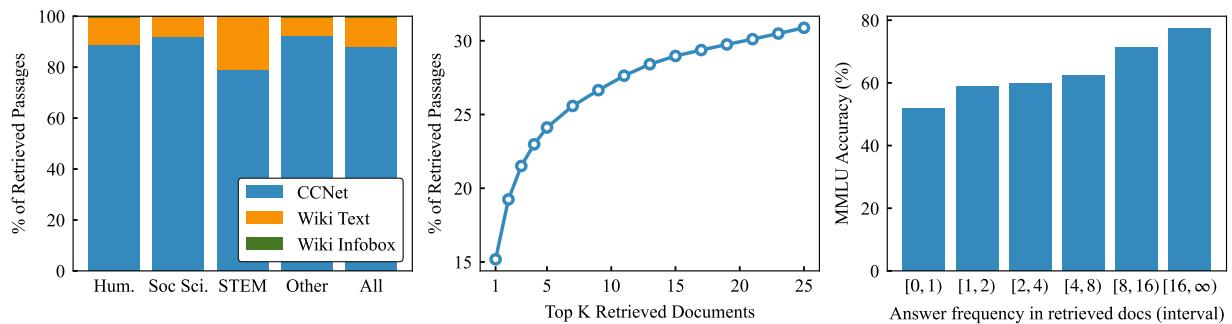


Figure 3: **MMLU Retrieval Analysis.** **Left:** Fraction of sources of top 30 retrieved passages for MMLU from CCNet, Wikipedia passages and info boxes for the 5-shot multitask ATLAS. **Center:** How often the text of the correct MMLU answer option appears in retrieved passages, as a function of the number of retrieved passages. **Right:** MMLU accuracy as a function of answer occurrence frequency in retrieved passages set

## 5 Analysis

### 5.1 Interpretability and Leakage

An advantage of semi-parametric models like ATLAS is the ability to inspect retrieved items to aid interpretability. To better understand how well ATLAS retrieves, and how it uses retrieved passages, we examine the retrieved passages for multi-task few-shot MMLU. As shown in the left panel of Figure 3, the model retrieves the majority of its passages from CCNet (85% on average). Wikipedia makes up about 15% of retrieved passages, which is higher than we would expect under a uniform prior, given Wikipedia only makes up about 10% of the index. The fraction of Wikipedia retrieval varies between MMLU domains, with the model using Wikipedia to a greater extent for STEM domains, and least for social sciences. The domain making the greatest use of Wikipedia is “abstract algebra” (73%), and the least is “moral scenarios” (3%). We also note that the MMLU-finetuned ATLAS does not make significant use of Wikipedia infobox passages.

We can also analyse the content of passages to assess how they may useful for accomplishing the downstream task. The middle panel of Figure 3 shows how often retrieved documents contain the text of the correct answer option. There being at least one mention of the correct answer choice in 30% of test questions in the top 25 passages.<sup>6</sup> The right panel shows that the accuracy on MMLU increases when the correct answer option text occurs more frequently in retrieved passages, rising from 55% for questions when the answer option does not appear, to 77% for questions mentioned more than 15 times.

A human analysis of retrieved documents revealed that documents are helpful for answering questions in a number of different ways. Manual inspection of a sample of 50 correctly-answered questions revealed that 44% contained at least partially useful background information. These are documents that would improve the likelihood of a non-expert human answering correctly, such as contextual clues surrounding a quotation from a question, or helpful numerical figures for quantity-based questions, which help to narrow down the answer options to a smaller range. In a further 26% of cases, a passage contained all the necessary information to answer the question, stated in a straightforward way. If read competently, such passages make the question simple to answer, and often include information such as canonical definitions, or the exact numerical answer requested in the question. 28% of retrieval sets did not contain obvious information which would make the question easier. Finally, 2% contained the verbatim question in a passage, together with its answer.

Given that MMLU has been created from pre-existing exams, it is possible that these questions appear on the open web. Models trained on web data (or, in our case, retrieving from it) run the risk of answering correctly not through generalisation, but by verbatim memorisation, which could lead to misleadingly high scores. In some very large language models, which can verbatim memorize and recall large parts of their pre-training

<sup>6</sup>Note: Depending on the question, it may not be important or useful to retrieve the exact text of the answer in MMLU, and as such, a hits@k value of 30% does not imply that retrieval fails to surface useful information in 70% of cases

---

data (Carlini et al., 2021), efforts have sometimes been made to filter occurrences of downstream instances from pre-training data, but this has not been performed for MMLU in the literature. In order to assess the prevalence of MMLU leakage in our index, we manually checked retrieval results for questions where the longest n-gram overlap between the question (without answer options) and a passage was at least 75% the length of the question. This resulted in an estimate of leakage of 2.8% of questions from our CC-Net corpus.

A benefit of retrieval-augmented models such as ATLAS is the editability of its knowledge (see section 5.2 for additional analysis). To estimate pure, non-leaked performance, we can filter out any potentially-leaked passages from retrieved results and rerun the language model. The MMLU score drops slightly when controlling for this leakage from 56.4 to 55.8% (-.5%). We note that our CC-net corpus is relatively small compared to the pre-trained corpora of recent very large models, which are trained on up to 1.4 trillion tokens (Hoffmann et al., 2022), 35x the size of our index, making it likely that models trained on corpora of that size would observe more MMLU leaked examples, but detecting such leakage is challenging in non-retrieval augmented models.

## 5.2 Temporal Sensitivity and Updateability

A benefit of retrieval-augmented models is that they can be kept up-to-date without retraining, by updating or swapping their index at test time. To assess the effectiveness of this mechanism in ATLAS, we first construct a dataset of time-sensitive questions derived from TempLAMA (Dhingra et al., 2022). TempLAMA is a collection of templated cloze questions derived from Wikidata and Wikidata where the correct answer changes over time. We select a subset of questions from this dataset which have a different answer in 2017 and 2020, for example, Question: *Theo Walcott plays for \_\_\_\_\_* Answer: *Arsenal F.C. (2017), Everton F.C. (2020)*, and form a small training set of 248 training, 112 development and 806 test questions.

Using this dataset, we finetune closed-book T5-XXL and ATLAS using the questions and the 2017 answers, supplying ATLAS with a 2017 Wikipedia index, and then measure exact match accuracy on the 2017 test set. The results can be found in the first row and first two columns of Table 11. We first observe that, as expected, ATLAS greatly outperforms T5 (57.7% c.f. 12.1%). We also note that, as desired, both T5 and ATLAS almost never generate an answer from 2020 when trained with the 2017 answers, scoring 2.8% and 1.5% respectively (first row, second two columns of Table 11). However, as shown in row 2, we can swap the ATLAS index to a 2020 Wikipedia index, *without retraining*, and find that ATLAS updates its predictions accordingly, with 2020 accuracy rising to a similar level to its 2017 performance (53.1%), whereas the purely parametric T5 has no such updateability mechanism.

This demonstrates that ATLAS can be faithful and condition strongly on its supplied index. Furthermore, this zero-shot updateability mechanism has the useful property of staying up-to-date without requiring up-to-date annotated data, or continuous, lifelong pre-training, as would be required for a large parametric-only model. Rows 3 and 4 of Table 11 complete the picture, where this time we train with 2020 answers, and demonstrate ATLAS can zero-shot transfer backwards in time to 2017 effectively too (50.1%). Interestingly, T5 is unable answer questions from 2020 well, even when trained with 2020 answers (3.6%), likely because it was pre-trained on data pre-dating 2020 (Dodge et al., 2021).

We also examine temporal effects for NaturalQuestions. NaturalQuestions is a dataset composed of search queries collected via the Google search engine in a short period of time. Thus data have a strong temporal bias, with a lot of questions about the 2018 World Cup for example. Moreover some questions are ambiguous without specification of the temporal context. For instance, for the question “*when did ireland last beat england at twickenham*”, the expected answer is 2018 in NaturalQuestions, while Ireland also beat England at Twickenham in 2022 as well as many other times before. In Table 12, we report results obtained by finetuning ATLAS using different Wikipedia dumps for the index. We observe that the 2018 December Wikipedia dump, which is close to the date of data collection, leads to the best results for both few-shot and full fine-tuning. In particular, it leads to a new state-of-the-art of 64 EM on NaturalQuestions.

Table 11: **Results on our TempLAMA-derived dataset.** We report performance for a static, closed-book T5-11B, as well as ATLAS-11B supplied with a test-time Wikipedia index from 2017 or 2020. We evaluate models finetuned on a small training set of 248 time-sensitive cloze-question-answer pairs, using answers either from 2017 or 2020. Good models should score highly when the test set year matches the year of the test-time index, and score low otherwise.

Train Set	Test-time Index	2017 Test Set Acc.		2020 Test Set Acc.	
		Closed-book	ATLAS	Closed-book	ATLAS
2017 answers	2017	12.1	57.7	2.9	1.5
	2020	12.1	10.2	2.9	53.1
2020 answers	2017	4.8	50.1	3.6	4.2
	2020	4.8	3.5	3.6	60.5

Table 12: **Impact of index data temporality on NaturalQuestions.** We report exact match performance on NaturalQuestions using different Wikipedia dumps in the index. We observe that the dump from December 2018, commonly used for NaturalQuestions, leads to the best result.

	Dec. 2017	Dec. 2018	Aug. 2019	Dec. 2020	Dec. 2021
64-shot	44.7	<b>45.1</b>	44.1	44.0	41.3
Full	63.2	<b>64.0</b>	62.4	61.1	59.6

### 5.2.1 Index Compression

Maintaining dense retrieval indices can be memory-intensive, especially as the number of indexed items is scaled. In this section, we briefly analyse the memory requirements of ATLAS’s index in the case of a) a Wikipedia index and b) the combined CC and Wikipedia index used in most of the experiments above.

There are two sources of memory pressure for ATLAS’s retrieval component – the passages themselves, and the document embedding index. The tokenized passages, once binarized, require 11GB and 130GB of storage for the Wikipedia and combined indices respectively. These passages do not need to be stored in expensive GPU RAM, and could even be memory-mapped to disk, sharded across nodes or compressed if required, and thus do not represent a limiting hardware challenge in this context. The embedding index itself, however, must be stored in GPU RAM for fast search, and thus its size is more sensitive. In the above experiments, we perform exact search over our index, which is achieved by sharding the index over all the the available GPUs, and computing the search in parallel. The index is stored at fp16 precision, resulting in a total GPU memory requirement of 49 GB and 587 GB for the Wikipedia and combined indices, respectively.

This large GPU memory requirement for the index limits accessibility and ease of deployment. However, many index compression techniques are available for nearest neighbour search, which can often dramatically reduce memory requirements at the cost of some retrieval accuracy. Following [Izacard et al. \(2020\)](#), we explore the effect of Product Quantization (PQ, [Jégou et al., 2011](#)), a popular lossy compression technique on ATLAS-3B’s accuracy for the 64-shot NQ task at different compression levels.

The results are shown in Figure 4. We find that substantial compression is possible before the onset of significant performance degradation. Namely, the Wikipedia index can be compressed from 49GB to 4GB with negligible drop in retrieval precision and exact match. Likewise, the combined index can be compressed from 587GB to 50GB without serious degradation, indicating that the combined index could be loaded onto a single 80GB GPU.

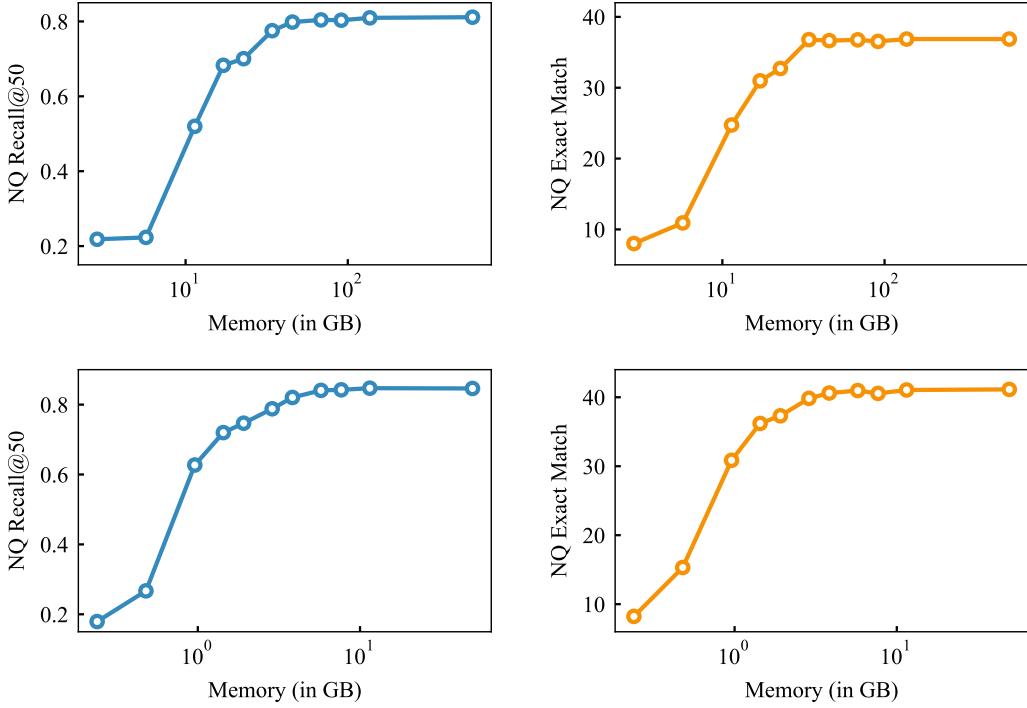


Figure 4: **Index Compression:** ATLAS-3B 64-shot NQ performance (left column: Retrieval Recall@50, right column: QA Exact Match score), as a function of index size, for different levels of quantisation. The right-most point in each plot represents the uncompressed index. **Top Row:** Wikipedia + CC Index. **Bottom Row:** Wikipedia Index.

## 6 Discussion

In this paper, we introduce ATLAS, a large retrieval-augmented language model. By jointly pre-training the retriever module and the language model, we show that ATLAS has strong few-shot learning capabilities on a wide range of knowledge intensive tasks, including NaturalQuestions, TriviaQA, FEVER, 8 KILT tasks and 57 MMLU tasks. For example, ATLAS-11B reaches more than 42% accuracy on NaturalQuestions and 84.7% on TriviaQA when training on 64 examples, which is an improvement of almost 3 points compared to PaLM, a 540B parameters model, which required 50x more pre-training compute. We also provided detailed ablations and analyses for what factors are important when training such retrieval-augmented models, and demonstrated ATLAS’s updateability, interpretability and controlability capabilities. Lastly, we demonstrated that ATLAS is also powerful in full-dataset settings obtaining a new state-of-the-art results on NaturalQuestions, TriviaQA, FEVER, and 5 KILT tasks.

## References

- Michele Bevilacqua, Giuseppe Ottaviano, Patrick Lewis, Wen-tau Yih, Sebastian Riedel, and Fabio Petroni. Autoregressive search engines: Generating substrings as document identifiers, 2022. URL <https://arxiv.org/abs/2204.10628>. 15
- Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens, 2021. URL <https://arxiv.org/abs/2112.04426>. 1, 8, 14

---

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL <https://arxiv.org/abs/2005.14165>. 1, 8, 14, 28

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. Autoregressive entity retrieval. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=5k8F6UU39V>. 15

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom B. Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *USENIX Security Symposium*, pp. 2633–2650, 2021. URL <https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting>. 17

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In *Proc. ACL*, 2017. 7

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022. URL <https://arxiv.org/abs/2204.02311>. 1, 2, 8, 14

Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. In *Proc. ACL*, 2018. 7

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018. 9

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL*, 2019. 7

Bhuwan Dhingra, Jeremy R. Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W. Cohen. Time-aware language models as temporal knowledge bases. *Transactions of the Association for Computational Linguistics*, 10:257–273, 2022. doi: 10.1162/tacl\_a\_00459. URL <https://aclanthology.org/2022.tacl-1.15>. 9, 17

Emily Dinan, Stephen Roller, Kurt Shuster, Angela Fan, Michael Auli, and Jason Weston. Wizard of wikipedia: Knowledge-powered conversational agents. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=r1173iRqKm>. 7, 9

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 1286–1305, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.98. URL <https://aclanthology.org/2021.emnlp-main.98>. 17

---

Hady Elsahar, Pavlos Vougiouklis, Arslan Remaci, Christophe Gravier, Jonathon Hare, Frederique Laforest, and Elena Simperl. T-REx: A large scale alignment of natural language with knowledge base triples. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018. URL <https://aclanthology.org/L18-1544>. 9

Martin Fajcik, Martin Docekal, Karel Ondrej, and Pavel Smrz. R2-D2: A modular baseline for open-domain question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 854–870, November 2021. doi: 10.18653/v1/2021.findings-emnlp.73. URL <https://aclanthology.org/2021.findings-emnlp.73>. 14

Michael Fink. Object classification from a single example utilizing class relevance metrics. In L. Saul, Y. Weiss, and L. Bottou (eds.), *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2005. URL <https://proceedings.neurips.cc/paper/2004/file/ef1e491a766ce3127556063d49bc2f98-Paper.pdf>. 8

Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3816–3830, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.295. URL <https://aclanthology.org/2021.acl-long.295>. 8

Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, Ankita Rajaram Naik, Pengshan Cai, and Alfio Gliozzo. Re2g: Retrieve, rerank, generate, 2022. URL <https://arxiv.org/abs/2207.06300>. 15

Edouard Grave, Moustapha Cisse, and Armand Joulin. Unbounded cache model for online language modeling with open vocabulary, 2017a. URL <https://arxiv.org/abs/1711.02604>. 8

Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. In *International Conference on Learning Representations*, 2017b. URL <https://openreview.net/forum?id=B184E5qee>. 8

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*, 2020. 1, 7

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020. 3

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. 9, 13, 28

Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust disambiguation of named entities in text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pp. 782–792, 2011. URL <https://aclanthology.org/D11-1072>. 9

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL <https://arxiv.org/abs/2203.15556>. 1, 6, 8, 14, 17, 28

Sebastian Hofstätter, Jiecao Chen, Karthik Raman, and Hamed Zamani. Multi-task retrieval-augmented text generation with relevance sampling. *ArXiv*, abs/2207.03030, 2022. 7, 15

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 2333–2338, 2013. 3, 7

---

Angeliki Lazaridou, Elena Gribovskaya, Wojciech Stokowiec, and Nikolai Grigorev. Internet-augmented language models through few-shot prompting for open-domain question answering. *ArXiv*, abs/2203.05115, 2022. 8, 14

Teven Le Scao and Alexander Rush. How many data points is a prompt worth? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2627–2636, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.208. URL <https://aclanthology.org/2021.naacl-main.208>. 8

Haejun Lee, Akhil Kedia, Jongwon Lee, Ashwin Paranjape, Christopher D. Manning, and Kyoung-Gu Woo. You only need one model for open-domain question answering, 2021a. URL <https://arxiv.org/abs/2112.07381>. 7

Jinhyuk Lee, Mujeen Sung, Jaewoo Kang, and Danqi Chen. Learning dense representations of phrases at scale. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 6634–6647, Online, August 2021b. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.518. URL <https://aclanthology.org/2021.acl-long.518>. 7

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. In *Proc. ACL*, 2019. 7

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *CoRR*, abs/2104.08691, 2021. URL <https://arxiv.org/abs/2104.08691>. 8

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pp. 333–342, 2017. doi: 10.18653/v1/K17-1034. URL <https://aclanthology.org/K17-1034>. 9

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401*, 2020. 1, 7, 14

Xiang Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL <https://aclanthology.org/2021.acl-long.353>. 8

Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. Jurassic-1: Technical details and evaluation. Technical report, AI21 Labs, August 2021. 8

IV Robert L. Logan, Ivana Balavzević, Eric Wallace, Fabio Petroni, Sameer Singh, and Sebastian Riedel. Cutting down on prompts and parameters: Simple few-shot learning with language models. *ArXiv*, abs/2106.13353, 2021. 8

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, October–November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL <https://aclanthology.org/D18-1260>. 9

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. Ambigqa: Answering ambiguous open-domain questions. *arXiv preprint arXiv:2004.10645*, 2020. 7

Reiichiro Nakano, Jacob Hilton, S. Arun Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. *ArXiv*, abs/2112.09332, 2021. 8

---

Ashwin Paranjape, Omar Khattab, Christopher Potts, Matei Zaharia, and Christopher D. Manning. Hindsight: Posterior-guided training of retrievers for improved open-ended generation, 2021. URL <https://arxiv.org/abs/2110.07752>. 7

Fabio Petroni, Patrick Lewis, Aleksandra Piktus, Tim Rocktäschel, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. How context affects language models' factual predictions. *arXiv preprint arXiv:2005.04611*, 2020. 9

Fabio Petroni, Samuel Broscheit, Aleksandra Piktus, Patrick Lewis, Gautier Izacard, Lucas Hosseini, Jane Dwivedi-Yu, Maria Lomeli, Timo Schick, Pierre-Emmanuel Mazaré, Armand Joulin, Edouard Grave, and Sebastian Riedel. Improving wikipedia verifiability with ai, 2022. URL <https://arxiv.org/abs/2207.06220>. 7

Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Dmytro Okhonko, Samuel Broscheit, Gautier Izacard, Patrick Lewis, Barlas Oğuz, Edouard Grave, Wen-tau Yih, and Sebastian Riedel. The web is your oyster - knowledge-intensive nlp against a very large web corpus, 2021. URL <https://arxiv.org/abs/2112.09924>. 15

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019. 8

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Ed Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. Scaling language models: Methods, analysis & insights from training gopher, 2021. URL <https://arxiv.org/abs/2112.11446>. 1, 8, 10, 14, 15

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019. 3, 5

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proc. EMNLP*, 2016. 7

Ori Ram, Gal Shachaf, Omer Levy, Jonathan Berant, and Amir Globerson. Learning to retrieve passages without supervision. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2687–2700, Seattle, United States, July 2022. Association for Computational Linguistics. URL <https://aclanthology.org/2022.naacl-main.193>. 7

Matthew Richardson. MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*, October 2013. URL <https://www.microsoft.com/en-us/research/publication/mctest-challenge-dataset-open-domain-machine-comprehension-text/>. 7, 9

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at TREC-3. *NIST Special Publication Sp*, 1995. 7

---

Devendra Singh Sachan, Siva Reddy, William Hamilton, Chris Dyer, and Dani Yogatama. End-to-end training of multi-document reader and retriever for open-domain question answering, 2021. URL <https://arxiv.org/abs/2106.05346>. 4, 5, 7

Timo Schick and H. Schütze. It's not just size that matters: Small language models are also few-shot learners. *ArXiv*, abs/2009.07118, 2021. 8

Timo Schick and Hinrich Schütze. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 255–269. Association for Computational Linguistics, 2021a. doi: 10.18653/v1/2021.eacl-main.20. URL <https://aclanthology.org/2021.eacl-main.20>. 8

Timo Schick and Hinrich Schütze. Few-shot text generation with natural language instructions. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 390–402. Association for Computational Linguistics, November 2021b. doi: 10.18653/v1/2021.emnlp-main.32. URL <https://aclanthology.org/2021.emnlp-main.32>. 8

Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*, pp. 373–374, 2014. doi: 10.1145/2567948.2577348. URL <https://doi.org/10.1145/2567948.2577348>. 7

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4222–4235, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.346. URL <https://aclanthology.org/2020.emnlp-main.346>. 8

Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. Retrieval augmentation reduces hallucination in conversation, 2021. URL <https://arxiv.org/abs/2104.07567>. 7

Kurt Shuster, Mojtaba Komeili, Leonard Adolphs, Stephen Roller, Arthur D. Szlam, and Jason Weston. Language models that seek for knowledge: Modular search & generation for dialogue and prompt completion. *ArXiv*, abs/2203.13224, 2022. 8

Shaden Smith, Mostafa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zhang, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model, 2022. URL <https://arxiv.org/abs/2201.11990>. 8

Derek Tam, Rakesh R Menon, Mohit Bansal, Shashank Srivastava, and Colin Raffel. Improving and simplifying pattern exploiting training. *ArXiv*, abs/2103.11955, 2021. 8

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam M. Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, Yaguang Li, Hongrae Lee, Huaixiu Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Yanqi Zhou, Chung-Ching Chang, I. A. Krivokon, Willard James Rusch, Marc Pickett, Kathleen S. Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Hartz Søraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Díaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravindran Rajakumar, Alena Butryna, Matthew Lamm, V. O. Kuzmina, Joseph Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. Lamda: Language models for dialog applications. *ArXiv*, abs/2201.08239, 2022. 8

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*, 2018. 7, 9

---

Sebastian Thrun and Lorien Pratt. *Learning to Learn: Introduction and Overview*, pp. 3–17. Kluwer Academic Publishers, USA, 1998. ISBN 0792380479. 8

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL <https://proceedings.neurips.cc/paper/2016/file/90e1357833654983612fb05e3ec9148c-Paper.pdf>. 8

Ellen M Voorhees et al. The TREC-8 question answering track report. In *TREC*, 1999. 7

Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. Multi-passage BERT: A globally normalized BERT model for open-domain question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 5878–5882, 2019. doi: 10.18653/v1/D19-1599. URL <https://aclanthology.org/D19-1599>. 7

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022. URL <https://arxiv.org/abs/2206.07682>. 8

Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. CCNet: Extracting high quality monolingual datasets from web crawl data. In *Proceedings of the 12th Language Resources and Evaluation Conference*, 2020. 10

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808*, 2020. 7

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL <https://arxiv.org/abs/1809.09600>. 9

Andrew Yates, Rodrigo Nogueira, and Jimmy Lin. Pretrained Transformers for Text Ranking: BERT and Beyond. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM ’21, pp. 1154–1156, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 978-1-4503-8297-7. doi: 10.1145/3437963.3441667. URL <https://doi.org/10.1145/3437963.3441667>. event-place: Virtual Event, Israel. 7

Wen-tau Yih, Kristina Toutanova, John C. Platt, and Christopher Meek. Learning Discriminative Projections for Text Similarity Measures. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, CoNLL ’11, pp. 247–256, USA, 2011. Association for Computational Linguistics. ISBN 978-1-932432-92-3. event-place: Portland, Oregon. 7

Dani Yogatama, Cyprien de Masson d’Autume, and Lingpeng Kong. Adaptive semiparametric language models. *Transactions of the Association for Computational Linguistics*, 9:362–373, 2021. doi: 10.1162/tacl\_a\_00371. URL <https://aclanthology.org/2021.tacl-1.22>. 1

---

Table 13: MMLU scores with de-biasing:

Setting	Model	All	Hum.	Soc. Sci.	STEM	Other
zero-shot	Standard	36.8	37.5	39.0	30.2	39.7
	All permutations	48.5	45.7	55.2	39.4	54.4
	Cyclic Permutations	47.1	43.6	54.1	38.0	54.9
5-shot	Standard	43.4	41.8	49.3	33.9	48.8
	All permutations	49.0	46.0	56.1	40.5	54.6
	Cyclic Permutations	47.9	46.1	54.6	38.8	52.8

## A Training details and additional results

### A.1 MMLU

#### A.1.1 Training Details

**Featurization** MMLU consists of multiple choice questions with four possible lexicalized answer options. We represent the input using the following template:

```
question: {question text}
options: (A) {answer 1 text} (B) {answer 2 text} (C) {answer 3 text} (D) {answer 4 text}
answer: [MASK_0]
```

and train the model to generate the mask token followed by the letter of the correct answer:

```
[MASK_0] {correct answer option letter}
```

This format closely matches the format of MLM pre-training objective, aiding few-shot learning. When training, we permute the order of the answer options, i.e. shuffling which answer option appears as letter A etc. This helps reduce overfitting, and encourages a uniform prior on the letters.

**Standard inference** Once trained we obtain predictions from the model by selecting the pre-softmax logits for the tokens A, B, C and D, and performing a softmax over them to obtain a distribution over the 4 answer options. For standard inference, we then simply return the answer corresponding to the argmax of this distribution.

**De-biased Inference** As mentioned in the main text, even though our model is finetuned with data that encourages a uniform prior over answer letters (by permuting which answer option letter is used with which lexical answer option text in training data), this may not be enough to ensure the model has no residual bias towards specific letters. Consider answers  $a$ , questions  $q$  and a nuisance variable  $z \in \mathcal{Z}$ , which represents the ordering of the answer options or, equivalently, which answer letter gets assigned to which answer option text. There are 4 answer options in MMLU, and thus  $|\mathcal{Z}| = 24$  unique ways they can be ordered, or assigned to given letters. Running our model with our standard inference for a question  $q$ , corresponds to calculating  $p(a|q = q, z = z)$  for the answer ordering  $z$  that happens to appear in the dataset. We can control for  $z$  by running the model with all possible answer orderings in the input, and marginalizing:  $p(a|q = q) = \sum_{z' \in \mathcal{Z}} p(a|q = q, z = z')p(z = z'|q = q)$ , and assuming  $p(z = z'|q = q)$  is uniform (no answer ordering is more likely than another), this reduces to simply  $p(a|q = q) \propto \sum_{z' \in \mathcal{Z}} p(a|q = q, z = z')$ . This procedure requires 24 forward passes, one for each answer ordering, so is  $24 \times$  slower than standard inference. Table 13 shows the result of applying the full permutation de-biasing, which leads to an 12% improvement zero-shot and 6% in 5-shot performance overall. Empirically, using only the cyclic permutations of the answer order provided in the original dataset (of which there are 4) works nearly as well, which is what we report in the main paper, and only increases inference compute by a factor of 4, rather than 24. Cyclic permutation de-biasing improves over standard inference by 10% in zero-shot and 5% in 5-shot. Empirically, de-biased inference is largely unnecessary when training in the 5-shot multitask or full dataset setting, as there is enough data for the model to learn a more uniform prior over the letters.

---

Table 14: Hyperparameters for MMLU

	770M	3B	11B
Batch size	64	64	64
Learning rate	(5e-5, 1e-5)	(5e-5, 1e-5)	(5e-5, 1e-5)
Retriever Temperature	0.1	0.1	0.1
5-shot train steps	64	32	16
5-shot (multitask) max train steps	2000	500	250
Full / transfer max train steps	5000	2000	2000

**Evaluation** We evaluate by following the method of [Hendrycks et al. \(2021\)](#), namely, micro-averaging across all 57 domains to obtain overall accuracy. We quote the results of GPT3 ([Brown et al., 2020](#)) and UnifiedQA ([Khashabi et al., 2020](#)) from the MMLU leaderboard at <https://github.com/hendrycks/test>. For Chinchilla and Gopher, we calculate the scores on the categories using the full MMLU results from [Hoffmann et al. \(2022\)](#).

**Index** The index used for MMLU for all MMLU experiments in the main paper comprised of concatenation of the Wikipedia passages, Wikipedia info boxes and Common Crawl indices, for a total of 387M passages. We can assess the importance of the index by running a model without the common crawl data, leading to a 5-shot multitask result of 52.8%, compared to 56.4% for the full model, a drop of 3.6%. This indicates that whilst the Wikipedia data is sufficient do well on the task, the addition of the CC data improves results further.

**Hyperparameters and development data** Selecting hyperparameters is challenging in few-shot settings. We do not assume access to an in-domain development set for the 5-shot task. Instead, we determine a set of hyperparameters for the 5-shot task using data from RACE, one of the auxiliary datasets provided by MMLU. Here, we sample 5 sets of 5-shot training data, and for each model size, we explore batch size {32, 64}, learning rates for the language model and retriever {(5e-5, 1e-5), (4e-5, 4e-5)}, retriever temperature {0.1, 0.01} and a fixed number of training steps {16, 32, 64, 128}, picking the setting that achieves strongest RACE validation scores. Having determined these hyperparameters, we apply them directly to the 5-shot MMLU task. For the 5-shot multi-task and full/transfer settings, we use the same batch size, temperatures and learning rates as the 5-shot task, but use a set of 285 MMLU validation examples (5 per domain) in order to determine the total number of training steps and for early stopping. The hyperparameters selected in the MMLU experiments can be found in table 14. We use query-side finetuning for the 5-shot and 5-shot multitask settings, and top-128 reranking for the full setting. For all MMLU runs we retrieve 30 documents

**Inter-run Variance** few-shot learning is well-known to suffer from high variance. In the main paper, we quote the result obtained with our first run. In order to assess the effect of noise and variance, we ran the 5-shot experiment with ATLAS 5 times.<sup>7</sup> We observe high variance for individual domains, sometimes as high as 20%, however, once aggregated across all 57 domains, the inter-run variance is low. The overall scores for these different runs, when using the same hyperparameters are shown in table 15. Due the effects of averaging over the many domains that comprise MMLU, the inter-run variance is quite modest on the aggregated metrics, with a std deviation of 0.5 in this experiment.

**Closed-Book Baselines** The closed book baselines we compare ATLAS to in table 5 are initialized from the same T5 model as their respective ATLAS, and then pre-trained with MLM for the same number of steps (10K) using the same pre-training data as ATLAS, for fairer comparison. The same procedure as for ATLAS was used to determine hyperparameters for MMLU for the closed-book models.

---

<sup>7</sup>This experiment was performed with a slightly different index to the main experiments, which achieves a stronger result

---

Table 15: Interrun Variance for 5-shot MMLU using ATLAS-11B

Run #	All	Hum.	Soc.	Sci.	STEM	Other
1	45.2	40.6	54.1	37.1	51.1	
2	45.1	39.8	54.4	37.1	52.0	
3	45.0	40.0	54.1	37.7	51.1	
4	45.6	41.3	54.7	37.0	51.6	
5	44.3	40.6	50.7	38.1	49.8	
Ave:	$45.0 \pm 0.5$	$40.5 \pm 0.6$	$53.6 \pm 1.6$	$37.4 \pm 0.5$	$51.1 \pm 0.8$	

### A.1.2 Full results

Tables 16 and 17 shows the full MMLU scores for each domain for ATLAS and the closed book T5 respectively. The full results for the cyclic-permutation-de-biased ATLAS-XXL can be found in Table 18.

## A.2 Question answering

### A.2.1 Training Details

For question answering, similarly to the MMLU experiments, we format the input using the following template:

```
question: {question text} answer: [MASK_0]
```

and train the model to generate the mask token followed by the answer:

```
[MASK_0] {answer}.
```

We generate answers using greedy decoding. For both training and testing, we retrieve 40 passages, and truncate the result of the concatenation between the query and the passages to 384 tokens.

For few-shot fine-tuning we train ATLAS for 30 steps using 64 random samples from the train sets. The retriever is trained using query-side fine-tuning. We select the model after 30 training steps. We use AdamW with a batch size of 32 and a learning rate of  $4 \times 10^{-5}$  with linear decay and 5 iterations of warmup for both the language model and the retriever.

For the fine-tuning on the full datasets, we train the model for 5k gradient steps and refresh the index every 500 steps for the first 1,000 training steps and every 2k training steps afterwards. We use AdamW with a batch size of 64 and a learning rate of  $4 \times 10^{-5}$  with linear decay and 5 iterations of warmup for both the language model and the retriever. We evaluate models every 500 steps and select the best one on the validation set based on the exact match score.

### A.2.2 Impact of scaling

In Table 19, we report performance on NaturalQuestions and TriviaQA as a function of the number of parameters in the reader module. Both for few-shot learning and full fine-tuning we observe strong improvements by scaling the size of the reader module. However we can notice sign of saturation when finetuning on full datasets, with limited gains when scaling from 3B to 11B parameters (+0.6% on NaturalQuestions, +0.5% on TriviaQA). While performance improves substantially when scaling from 3B to 11B parameters with 64 training samples, with +3.7% and +1.2% improvement on NaturalQuestions and TriviaQA respectively. For these experiments we use a setup similar to the one use in Table 8, except that we use an index composed of the December 2018 Wikipedia dump processed as described in section 4.2.

## A.3 KILT

For the results on KILT reported in Table 10 we fine-tune ATLAS individually on each dataset. We format the input using a template similar to the one used for question answering:

	5-shot			5-shot (multi-task)			Full / Transfer		
	770M	3B	11B	770M	3B	11B	770M	3B	11B
All	38.9	42.3	43.4	42.1	48.7	56.4	56.3	59.9	65.8
Humanities	37.3	40.0	41.9	37.7	46.4	50.0	50.9	53.0	60.3
Social Sciences	41.7	46.8	49.3	47.5	53.7	65.6	66.0	70.8	77.2
STEM	32.3	35.0	33.9	34.4	39.4	46.2	44.8	50.7	53.4
Other	44.9	48.1	48.8	50.4	55.9	66.6	65.5	68.1	74.4
abstract algebra	30.0	27.0	28.0	27.0	31.0	30.0	22.0	27.0	33.0
anatomy	28.9	50.4	45.2	44.4	57.8	64.4	57.8	68.9	69.6
astronomy	55.3	59.9	59.2	52.6	66.4	67.8	69.1	78.3	79.6
business ethics	49.0	51.0	48.0	50.0	62.0	60.0	51.0	70.0	68.0
clinical knowledge	41.9	44.9	40.0	46.8	54.3	64.9	64.2	72.5	74.0
college biology	38.2	45.8	50.0	36.8	52.1	63.2	63.2	72.2	78.5
college chemistry	32.0	29.0	29.0	31.0	33.0	38.0	45.0	39.0	45.0
college computer science	33.0	35.0	30.0	23.0	29.0	30.0	43.0	48.0	47.0
college mathematics	31.0	31.0	28.0	29.0	27.0	34.0	32.0	29.0	36.0
college medicine	31.2	35.8	38.2	50.3	40.5	52.0	60.1	59.5	63.6
college physics	20.6	26.5	31.4	21.6	28.4	39.2	27.5	44.1	42.2
computer security	53.0	50.0	55.0	49.0	61.0	64.0	69.0	71.0	76.0
conceptual physics	34.9	41.7	37.4	40.9	43.4	57.0	53.2	58.3	59.6
econometrics	28.9	21.1	27.2	26.3	25.4	34.2	28.9	37.7	36.8
electrical engineering	26.9	31.7	31.7	38.6	44.1	51.7	61.4	60.7	67.6
elementary mathematics	25.9	28.8	29.4	29.6	30.2	32.8	29.6	35.5	33.9
formal logic	34.9	33.3	33.3	23.0	30.2	29.4	34.1	38.9	34.1
global facts	28.0	34.0	34.0	36.0	40.0	49.0	50.0	49.0	52.0
high school biology	24.8	37.7	27.7	48.7	57.1	66.5	66.5	76.8	81.9
high school chemistry	34.5	31.0	31.0	31.5	36.5	48.3	44.8	52.2	52.2
high school computer science	31.0	39.0	28.0	37.0	42.0	42.0	50.0	59.0	57.0
high school european history	42.4	49.7	53.3	50.9	58.2	69.7	70.9	73.9	80.0
high school geography	38.9	42.4	50.0	46.5	56.6	69.2	74.2	80.8	82.8
high school gov. and pol.	57.5	60.6	60.1	52.9	64.8	76.7	80.8	85.5	91.7
high school macroeconomics	32.8	39.7	44.9	39.0	45.6	57.2	55.1	63.1	66.7
high school mathematics	30.7	33.0	35.6	28.1	27.8	37.0	30.7	34.8	37.0
high school microeconomics	34.5	42.9	45.4	44.1	51.7	68.9	63.4	70.2	81.1
high school physics	18.5	24.5	22.5	25.8	25.8	33.1	27.2	30.5	39.7
high school psychology	52.8	61.1	59.8	56.7	67.2	79.4	76.3	84.0	87.0
high school statistics	39.8	29.6	34.7	27.3	34.7	38.0	37.0	43.1	45.8
high school us history	43.6	49.0	55.9	46.1	57.8	59.8	62.7	72.5	76.5
high school world history	48.1	52.7	59.9	48.1	66.2	65.4	70.0	78.5	79.7
human aging	46.2	44.8	39.5	48.0	55.2	60.1	56.1	68.2	73.1
human sexuality	41.2	43.5	27.5	46.6	51.1	59.5	77.1	72.5	81.7
international law	54.5	57.9	60.3	55.4	72.7	73.6	81.8	82.6	85.1
jurisprudence	38.9	55.6	32.4	53.7	60.2	73.1	76.9	73.1	81.5
logical fallacies	43.6	54.0	57.1	44.2	58.3	70.6	64.4	73.0	76.7
machine learning	36.6	34.8	28.6	31.3	37.5	46.4	36.6	47.3	50.9
management	45.6	51.5	52.4	48.5	52.4	81.6	78.6	75.7	87.4
marketing	59.4	67.1	70.5	66.7	74.4	83.8	83.8	83.3	91.9
medical genetics	50.0	53.0	58.0	56.0	61.0	75.0	68.0	78.0	81.0
miscellaneous	63.0	64.2	68.8	64.0	72.4	84.3	85.4	83.9	90.9
moral disputes	37.0	41.3	41.3	40.8	50.3	60.1	61.9	66.2	73.7
moral scenarios	24.7	24.7	26.5	21.9	26.9	26.6	23.8	23.8	35.8
nutrition	40.9	45.1	45.1	49.0	52.3	67.0	64.7	68.6	76.8
philosophy	48.6	50.5	56.3	49.8	59.2	69.5	70.4	73.0	77.8
prehistory	45.7	50.0	52.8	54.9	64.8	74.4	69.8	75.0	80.6
professional accounting	28.4	33.0	34.0	35.1	34.0	45.7	43.6	46.1	51.8
professional law	32.4	33.5	34.8	30.4	37.6	39.1	41.5	41.5	50.5
professional medicine	29.4	26.1	27.6	34.6	40.8	52.2	47.8	43.4	59.6
professional psychology	37.7	43.0	50.2	45.1	51.0	60.6	59.5	62.4	74.0
public relations	40.0	46.4	44.5	51.8	54.5	66.4	63.6	66.4	68.2
security studies	35.1	33.5	38.8	44.1	39.6	57.6	60.8	61.6	72.2
sociology	45.3	51.2	51.2	52.7	60.2	69.2	74.1	78.6	85.1
us foreign policy	58.0	70.0	73.0	63.0	63.0	74.0	80.0	80.0	83.0
virology	34.3	34.3	32.5	38.0	42.8	45.2	47.6	49.4	53.0
world religions	65.5	69.0	71.9	70.2	82.5	80.1	83.6	83.6	87.1

Table 16: MMLU Test set scores for ATLAS for each model size and each of the 57 domains.

	5-shot			5-shot (multi-task)			Full / Transfer		
	770M	3B	11B	770M	3B	11B	770M	3B	11B
All	29.2	35.7	36.1	26.5	40.0	43.5	42.4	50.4	54.0
Humanities	30.5	35.4	35.5	27.3	38.5	41.6	41.0	48.6	51.3
Social Sciences	29.7	38.0	39.4	24.8	43.8	48.9	48.6	57.8	64.7
STEM	29.0	31.4	30.8	26.5	32.8	35.8	33.4	40.6	41.7
Other	26.7	37.7	38.6	27.0	45.0	48.5	46.8	55.2	59.1
abstract algebra	26.0	23.0	21.0	29.0	30.0	26.0	23.0	29.0	26.0
anatomy	21.5	40.0	40.7	27.4	39.3	45.9	35.6	43.7	42.2
astronomy	37.5	38.8	37.5	27.6	39.5	41.4	36.2	50.7	55.3
business ethics	29.0	54.0	42.0	26.0	47.0	55.0	53.0	64.0	60.0
clinical knowledge	32.5	33.6	40.0	28.7	44.2	47.9	45.3	52.8	57.7
college biology	29.9	34.7	34.0	29.9	34.7	40.3	38.2	46.5	52.1
college chemistry	37.0	22.0	32.0	20.0	35.0	33.0	36.0	34.0	36.0
college computer science	28.0	35.0	34.0	28.0	27.0	36.0	31.0	44.0	35.0
college mathematics	31.0	29.0	27.0	22.0	34.0	27.0	30.0	33.0	32.0
college medicine	24.3	34.7	34.1	27.2	40.5	40.5	35.8	41.6	48.6
college physics	33.3	23.5	23.5	22.5	19.6	26.5	22.5	32.4	24.5
computer security	36.0	42.0	46.0	31.0	49.0	52.0	50.0	65.0	61.0
conceptual physics	26.4	35.7	30.2	23.4	30.6	32.8	34.5	37.4	43.8
econometrics	26.3	21.9	28.9	17.5	19.3	24.6	29.8	25.4	29.8
electrical engineering	31.0	33.1	31.7	31.0	31.0	36.6	41.4	47.6	51.7
elementary mathematics	26.2	27.5	28.0	27.0	31.2	33.3	25.9	31.2	35.5
formal logic	34.1	34.1	31.7	15.1	34.9	31.0	31.7	38.1	42.1
global facts	32.0	30.0	25.0	34.0	34.0	27.0	28.0	34.0	30.0
high school biology	22.6	31.9	29.7	27.1	41.6	50.0	43.5	57.7	60.6
high school chemistry	27.1	26.6	27.6	28.6	31.5	29.1	30.5	36.5	38.9
high school computer science	26.0	32.0	25.0	33.0	37.0	45.0	45.0	55.0	48.0
high school european history	34.5	43.0	42.4	24.2	60.0	59.4	58.2	69.1	76.4
high school geography	31.3	40.4	36.9	24.7	45.5	50.5	56.1	66.7	74.2
high school gov. and pol.	28.0	49.2	51.3	19.2	56.0	59.6	55.4	70.5	75.6
high school macroeconomics	25.6	37.7	32.1	26.7	42.3	43.6	41.0	51.5	56.4
high school mathematics	35.9	35.2	35.9	28.1	26.7	31.1	27.8	36.7	31.9
high school microeconomics	27.3	29.8	36.1	20.6	35.7	42.9	42.9	50.8	60.5
high school physics	21.9	25.2	22.5	24.5	28.5	29.1	27.8	31.1	27.8
high school psychology	26.1	46.4	51.0	24.8	54.3	60.2	56.3	67.3	76.1
high school statistics	27.8	33.3	33.3	17.6	30.6	33.8	32.9	33.3	37.0
high school us history	30.4	39.7	45.6	27.5	46.1	58.3	51.0	63.2	72.5
high school world history	42.6	50.6	41.8	29.1	54.0	64.6	66.7	72.2	73.8
human aging	28.3	37.2	29.6	26.0	45.3	46.2	46.6	57.0	62.8
human sexuality	29.8	34.4	41.2	25.2	42.0	44.3	51.1	58.0	59.5
international law	57.9	57.9	41.3	44.6	57.9	58.7	62.8	71.9	71.1
jurisprudence	30.6	33.3	34.3	32.4	49.1	52.8	55.6	67.6	74.1
logical fallacies	40.5	55.8	46.6	25.8	51.5	62.0	43.6	69.3	71.2
machine learning	33.0	34.8	36.6	29.5	35.7	37.5	32.1	37.5	42.9
management	21.4	29.1	40.8	24.3	47.6	50.5	60.2	69.9	70.9
marketing	38.9	58.5	60.7	31.2	67.9	75.6	69.2	79.9	85.9
medical genetics	26.0	36.0	36.0	29.0	43.0	44.0	40.0	54.0	50.0
miscellaneous	24.5	45.2	46.4	27.1	52.2	58.2	51.3	64.6	72.7
moral disputes	32.4	37.3	38.7	28.6	43.4	43.4	49.7	64.7	64.7
moral scenarios	24.7	24.7	24.7	23.0	23.9	24.7	23.8	24.0	23.8
nutrition	30.1	33.0	34.6	25.8	42.5	44.1	50.3	55.6	61.1
philosophy	28.6	32.5	37.3	31.2	38.9	45.0	44.1	56.6	59.2
prehistory	33.6	37.0	41.4	27.5	39.8	50.6	41.0	51.5	57.7
professional accounting	21.3	28.0	30.5	25.9	35.5	34.0	37.2	41.5	42.2
professional law	28.2	33.4	34.0	27.6	35.4	35.5	38.3	43.0	45.6
professional medicine	19.5	26.5	24.3	20.2	32.0	37.9	38.6	40.8	46.0
professional psychology	27.8	32.8	32.8	26.6	39.5	43.6	38.4	48.0	58.3
public relations	22.7	43.6	40.0	21.8	47.3	56.4	50.0	55.5	60.0
security studies	37.6	26.1	31.0	20.4	34.7	44.1	56.3	61.6	66.9
sociology	43.3	41.8	38.8	30.8	45.8	52.7	60.2	66.7	72.1
us foreign policy	49.0	57.0	66.0	38.0	56.0	61.0	59.0	75.0	76.0
virology	29.5	26.5	34.3	30.1	36.1	39.8	44.0	46.4	41.6
world religions	24.0	40.9	47.4	32.7	49.1	57.3	48.0	63.7	70.2

Table 17: MMLU Test scores for the T5 closed book baseline for each model size and each of the 57 domains.

Domain	zero-shot	5-shot	5-shot (multi-task)	Full / Transfer
All	47.1	47.9	56.6	66.0
Humanities	43.6	46.1	50.1	61.1
Social Sciences	54.1	54.6	66.4	77.2
STEM	38.0	38.8	46.4	53.2
Other	53.9	52.8	66.2	74.4
abstract algebra	22.0	26.0	31.0	31.0
anatomy	48.9	47.4	62.2	70.4
astronomy	61.8	62.5	68.4	81.6
business ethics	60.0	57.0	62.0	70.0
clinical knowledge	50.6	49.4	66.4	72.8
college biology	51.4	53.5	61.1	77.8
college chemistry	36.0	39.0	39.0	45.0
college computer science	32.0	32.0	33.0	49.0
college mathematics	30.0	35.0	35.0	34.0
college medicine	44.5	41.0	52.6	67.6
college physics	24.5	26.5	37.3	42.2
computer security	59.0	59.0	68.0	76.0
conceptual physics	37.0	41.3	57.0	60.0
econometrics	20.2	20.2	36.8	37.7
electrical engineering	37.9	40.0	50.3	65.5
elementary mathematics	31.2	28.0	30.7	36.5
formal logic	27.8	27.0	32.5	35.7
global facts	41.0	43.0	51.0	53.0
high school biology	53.2	56.5	68.7	83.2
high school chemistry	41.9	41.4	49.3	51.2
high school computer science	40.0	36.0	46.0	60.0
high school european history	56.4	58.8	68.5	80.6
high school geography	57.1	59.6	71.2	81.3
high school gov. and pol.	67.9	67.9	77.2	90.2
high school macroeconomics	46.9	48.5	57.9	65.9
high school mathematics	28.1	28.9	34.1	31.5
high school microeconomics	51.7	51.7	68.9	82.4
high school physics	26.5	25.8	32.5	41.1
high school psychology	66.2	65.5	78.9	86.8
high school statistics	31.5	30.1	43.1	45.8
high school us history	57.8	54.9	64.7	77.5
high school world history	59.1	62.9	65.4	79.3
human aging	48.4	50.7	60.5	70.4
human sexuality	55.7	54.2	61.8	84.0
international law	66.1	72.7	71.9	84.3
jurisprudence	61.1	64.8	72.2	81.5
logical fallacies	54.6	57.7	71.2	77.9
machine learning	37.5	39.3	43.8	44.6
management	56.3	56.3	79.6	89.3
marketing	72.2	73.1	84.6	91.9
medical genetics	55.0	58.0	71.0	81.0
miscellaneous	69.7	67.8	83.8	90.4
moral disputes	45.1	46.8	60.1	72.3
moral scenarios	24.5	30.3	25.8	38.5
nutrition	56.5	53.9	67.0	77.1
philosophy	56.3	57.6	70.7	77.2
prehistory	59.3	60.5	71.6	78.7
professional accounting	35.1	33.0	42.2	50.7
professional law	36.3	38.4	39.4	51.7
professional medicine	35.7	33.1	52.2	60.7
professional psychology	47.7	49.3	60.9	74.0
public relations	54.5	53.6	68.2	68.2
security studies	47.3	45.7	59.2	73.9
sociology	62.2	62.7	71.6	84.6
us foreign policy	64.0	68.0	73.0	83.0
virology	39.8	40.4	44.6	51.8
world religions	77.2	74.9	80.7	87.1

Table 18: MMLU Test set scores for the de-biased ATLAS-XXL using cyclic permutations for each of the 57 domains for zero-shot, 5 shot, 5-shot-multitask and the transfer setting.

Number of parameters	220M	770M	3B	11B
NaturalQuestions 64-shot	27.0	35.4	41.3	45.1
NaturalQuestions full	54.1	60.8	63.4	64.0
TriviaQA 64-shot	55.3	65.0	70.2	71.4
TriviaQA full	71.8	74.9	77.5	78.0

Table 19: **Impact of model size on question answering datasets.** We report exact match performance on the test sets of NaturalQuestions and TriviaQA filtered depending on the number of parameters in the reader module. For these experiments the index contains the December 2018 Wikipedia dump.

Model	AIDA	FEV	T-REx	zsRE	NQ	HoPo	TQA	WoW
	ACC	ACC	ACC	ACC	EM	EM	EM	F1
ATLAS 64-shot	69.0	88.1	58.5	60.2	44.2	34.1	77.1	15.4
ATLAS full dataset	92.7	94.4	84.8	80.9	63.4	51.4	84.4	21.0

Table 20: **Downstream results on KILT dev sets.**

question: {query text} answer: [MASK\_0]

and train the model to generate the mask token followed by the expected output:

[MASK\_0] {output}.

We retrieve 20 passages and generate answer using greedy decoding. In KILT, FEVER is a two-way classification task of claims. We lexicalize the “SUPPORTS” (resp. ‘REFUTES’) label into “true” (respectively “false”).

For few-shot fine-tuning we train ATLAS for 30 steps using 64 random samples from the train sets. The retriever is trained using query-side fine-tuning. We evaluate models every 5 steps and select the best one on the development set based on the reported metric. We use AdamW with a batch size of 32 and a learning rate of  $4 \times 10^{-5}$  with linear decay and 5 iterations of warmup for both the language model and the retriever.

For the fine-tuning on the full datasets, the model is trained for 5k gradient steps. We evaluate models every 500 steps and select the best one on the development set based on the reported metric. The index is refreshed every 500 step for the first 1000 iterations, and every 2k steps afterwards. We use AdamW with a batch size of 64 and a learning rate of  $4 \times 10^{-5}$  with linear decay and 500 iterations of warmup for both the language model and the retriever.

We report results on the development sets in Table 20.

# Retrieval as Attention: End-to-end Learning of Retrieval and Reading within a Single Transformer

Zhengbao Jiang<sup>♡\*</sup>, Luyu Gao<sup>♡\*</sup>, Jun Araki<sup>◊</sup>, Haibo Ding<sup>◊†</sup>,  
 Zhiruo Wang<sup>♡</sup>, Jamie Callan<sup>♡</sup>, Graham Neubig<sup>♡</sup>

<sup>♡</sup>Language Technologies Institute, Carnegie Mellon University

<sup>◊</sup>Bosch Research North America

{zhengbaj, luyug, zhiruow, callan, gneubig}@cs.cmu.edu

{jun.araki, haibo.ding}@us.bosch.com

## Abstract

Systems for knowledge-intensive tasks such as open-domain question answering (QA) usually consist of two stages: efficient retrieval of relevant documents from a large corpus and detailed reading of the selected documents to generate answers. Retrievers and readers are usually modeled separately, which necessitates a cumbersome implementation and is hard to train and adapt in an end-to-end fashion. In this paper, we revisit this design and eschew the separate architecture and training in favor of a single Transformer that performs **R**etrieval as **A**ttention (**ReAtt**), and end-to-end training solely based on supervision from the end QA task. We demonstrate for the first time that a single model trained end-to-end can achieve both competitive retrieval and QA performance, matching or slightly outperforming state-of-the-art separately trained retrievers and readers. Moreover, end-to-end adaptation significantly boosts its performance on out-of-domain datasets in both supervised and unsupervised settings, making our model a simple and adaptable solution for knowledge-intensive tasks. Code and models are available at <https://github.com/jzbjyb/ReAtt>.

## 1 Introduction

Knowledge-intensive tasks such as question answering (QA), fact checking, and dialogue generation require models to gather relevant information from potentially enormous knowledge corpora (e.g., Wikipedia) and generate answers based on gathered evidence. A widely used solution is to first *retrieve* a small number of relevant documents from the corpus with a *bi-encoder* architecture which encodes queries and documents independently for efficiency purposes, then *read* the retrieved documents in a more careful and expansive way with a *cross-encoder* architecture which encodes queries

and documents jointly (Lee et al., 2019; Guu et al., 2020; Lewis et al., 2020; Izacard et al., 2022). The distinction between retrieval and reading leads to the widely adopted paradigm of treating retrievers and readers separately. Retriever and readers are usually two separate models with heterogeneous architectures and different training recipes, which is cumbersome to train. Even though two models can be combined in an ad-hoc way for downstream tasks, it hinders effective end-to-end learning and adaptation to new domains.

There have been several attempts to connect up reader and retriever training (Lee et al., 2019; Guu et al., 2020; Lewis et al., 2020; Sachan et al., 2021; Lee et al., 2021a; Izacard et al., 2022). However, retrievers in these works are not learned in a fully end-to-end way. They require either initialization from existing supervisedly trained dense retrievers (Lewis et al., 2020), or expensive unsupervised retrieval pretraining as warm-up (Lee et al., 2019; Guu et al., 2020; Sachan et al., 2021; Lee et al., 2021a; Izacard et al., 2022). The reliance on retrieval-specific warm-up and the ad-hoc combination of retrievers and readers makes them less of a unified solution and potentially hinders their domain adaptation ability. With the ultimate goal of facilitating downstream tasks, retriever and reader should instead be fused more organically and learned in a fully end-to-end way.

In this paper, we focus on one of the most important knowledge-intensive tasks, open-domain QA. We ask the following question: is it possible to perform both retrieval and reading *within a single Transformer model*, and train the model in a *fully end-to-end* fashion to achieve competitive performance from both perspectives? Such a single-model end-to-end solution eliminates the need for retrieval-specific annotation and warm-up and simplifies retrieval-augmented training, making adaptation to new domains easier. Based on the analogy between self-attention which relates dif-

---

\*The first two authors contributed equally.

†Haibo Ding is now at Amazon.

ferent tokens in a single sequence (Vaswani et al., 2017) and the goal of retrieval which is to relate queries with relevant documents, we hypothesize that self-attention could be a natural fit for retrieval, and it allows an organic fusion of retriever and reader within a single Transformer.

Specifically, we start from a encode-decoder T5 (Raffel et al., 2020) and use it as both retriever and reader. We use the first  $B$  encoder layers as bi-encoder to encode queries and documents independently, and the attention score at layer  $B + 1$  (denoted as *retrieval attention*) to compute relevance scores, as shown in Fig. 1. We found that directly using self-attention for retrieval underperforms strong retrievers, which we conjecture is because self-attention pretrained on local context is not sufficient to identify relevant information in the large representation space of the whole corpus. To solve this, we propose to compute retrieval attention between a query and a large number of documents and *adjust the retrieval attention across documents*. For each query, we compute retrieval attention over both close documents that potentially contain positive and hard negative documents, and documents of other queries in the same batch as random negatives. The retrieval attention is adjusted by minimizing its discrepancy from the cross-attention between the decoder and encoder (denoted as *target attention*), which is indicative of the usefulness of each document in generating answers (Izacard and Grave, 2021a). The resulting **Retrieval as Attention model (ReAtt)** is a single T5 trained based on only QA annotations and simultaneously learns to promote useful documents through cross-document adjustment.

We train ReAtt on Natural Questions dataset (NQ) (Kwiatkowski et al., 2019) in a fully end-to-end manner. It achieves both competitive retrieval and QA performance, matching or slightly outperforming state-of-the-art retriever ColBERT-NQ (Khattab et al., 2020) trained with explicit retrieval annotations and strong QA model FiD (Izacard and Grave, 2021b,a), demonstrating for the first time end-to-end training can produce competitive retriever and reader within a single model. To further test ReAtt’s generalization and end-to-end adaptation ability, we conduct zero-shot, supervised, and unsupervised adaptation experiments on 7 datasets from the BEIR benchmark (Thakur et al., 2021). In all settings, end-to-end adaptation improves the retrieval performance usually by a large margin,

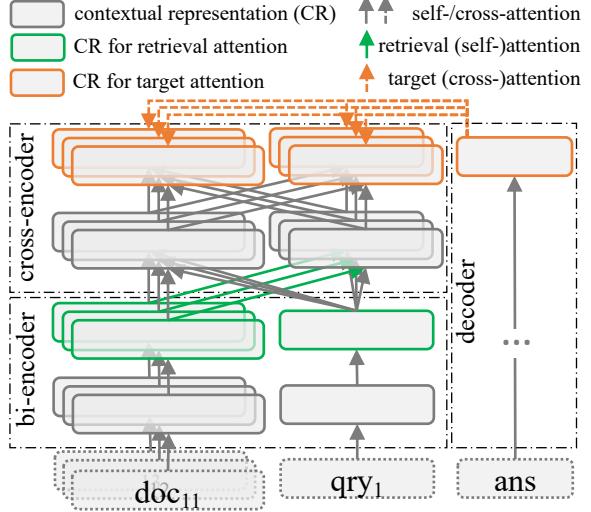


Figure 1: Illustration of Retrieval as Attention (ReAtt) with the first  $B=2$  encoder layers as bi-encoder (i.e., retriever) and the rest  $L-B=2$  layers as cross-encoder. During training, the **retrieval attention** between a query  $q_1$  and documents  $d_{11,12,13}$  is adjusted by minimizing its discrepancy from the **target attention**. For simplicity, we use a single arrow to represent attention of a single head between multiple tokens.

achieving comparable or superior performance to strong retrieval adaptation and pretraining methods.

## 2 Retrieval as Attention (ReAtt)

With the goal of developing a single Transformer that can perform both retrieval and reading, and the analogy between retrieval and self-attention, we first introduce architecture changes to allow retrieval as attention (§ 2.2), then examine how well attention as-is can be directly used to perform retrieval (§ 2.3).

### 2.1 Formal Definition

We first briefly define the task of retrieval and question answering. As mentioned in the introduction, queries and documents need to be represented independently for efficient retrieval which implies a bi-encoder architecture that has no interaction between queries and documents. Without loss of generality, we use  $E_d = \text{biencoder}(d)$  to denote one or multiple representations generated by a bi-encoder based on a document from a corpus  $d \in \mathcal{D}$ , and likewise  $E_q = \text{biencoder}(q)$  to denote query representations.<sup>1</sup> The top-k documents most relevant to a query are retrieved by  $\mathcal{D}_q^{\text{ret}} = \arg \text{topk}_{d \in \mathcal{D}} r(E_q, E_d)$ , where function  $r$

<sup>1</sup>Queries and documents can use different bi-encoders but we use one notation for simplicity.

computes relevance based on query and document representations which can be as simple as a dot product if queries and documents are encoded into a single vector, and  $\mathcal{D}_q^{\text{ret}}$  stands for the returned documents. We consider encoder-decoder-based generative question answering in this paper, which jointly represents queries and retrieved documents with the encoder  $E_{q,d} = \text{crossencoder}(q, d)$ , and generates the answer  $a$  autoregressively with the decoder  $P^{\text{gen}}(a|q, d) = P^{\text{gen}}(a|E_{q,d})$ . To handle multiple retrieved documents, we follow the fusion-in-decoder model (FiD) (Izacard and Grave, 2021b) which encodes each query-document pair independently and fuse these representations in decoder through cross-attention  $P^{\text{gen}}(a|q, \mathcal{D}_q^{\text{ret}}) = P^{\text{gen}}(a|E_{q,d_1}, \dots, E_{q,d_{|\mathcal{D}_q^{\text{ret}}|}})$ . Negative log likelihood (NLL) is used in optimization  $\mathcal{L}_{\text{QA}} = -\log P^{\text{gen}}(a|q, \mathcal{D}_q^{\text{ret}})$ .

## 2.2 Leveraging Attention for Retrieval

Next, we introduce our method that directly uses self-attention between queries and documents as retrieval scores.

**Putting the Retriever into Transformers** As illustrated in Fig. 1, we choose T5 (Raffel et al., 2020) as our base model, use the first  $B$  layers of the encoder as the *bi-encoder “retriever”* by disabling self-attention between queries and documents, and the remaining  $L - B$  layers as the *cross-encoder “reader”*. We use the self-attention paid from query tokens to document tokens at the  $B + 1$ -th layer as the retrieval score, which is denoted as *retrieval attention* (green arrows in Fig. 1). It is computed based on the independent query and document contextual representations from the last ( $B$ -th) layer of the bi-encoder (green blocks in Fig. 1). Formally for an  $H$ -head Transformer, document and query representations are:

$$\begin{aligned} E_d &= \{K_d^{B+1,h} \in \mathbb{R}^{|d| \times e}\}_{h=1}^H, \\ E_q &= \{Q_q^{B+1,h} \in \mathbb{R}^{|q| \times e}\}_{h=1}^H, \end{aligned}$$

where  $K$  and  $Q$  are key and query vectors of the token sequence used in self-attention,  $|d|$  and  $|q|$  are document and query length, and  $e$  is the dimensionality of each head. The retrieval attention matrix from query tokens to document before softmax for one head is computed by:

$$A_{q,d}^{B+1,h} = Q_q^{B+1,h} \times K_d^{B+1,hT} \in \mathbb{R}^{|q| \times |d|}.$$

Directly using attention for retrieval can not only leverage its ability to identify relatedness, it is also

a natural and simple way to achieve both retrieval and reading in a single Transformer with minimal architectural changes, which facilitates our final goal of end-to-end learning.

## From Token Attention to Document Relevance

Given the token-level attention scores  $A_{q,d}^{B+1,h}$ , the relevance between  $q$  and  $d$  is computed by avg-max aggregation: choosing the most relevant document token for each query token (i.e., max) then averaging across query tokens:

$$r_h(q, d) = \text{avg}_0(\max_1(A_{q,d}^{B+1,h})), \quad (1)$$

where 1 and 0 refer to the dimension over which the operation is applied. This is similar to the MaxSim and sum operators used in ColBERT (Khattab and Zaharia, 2020), with the intuition that a relevant document should match as many query tokens as possible with the best-matching token. The final relevance is a weighted sum over all heads:

$$r(q, d) = \sum_{h=1}^H P_h^{\text{head}} \cdot r_h(q, d),$$

where  $P_h$  is a learnable weight that sums to one. As explained in the next section, we empirically find only a few attention heads with non-random retrieval performance, and among them one particular head is significantly better than the others. Given this observation, we introduce a low temperature  $\tau$  to promote this sparsity  $P_h^{\text{head}} = \frac{\exp(w_h/\tau)}{\sum_{h'} \exp(w_{h'}/\tau)}$ , which always ends with *a single head* with the great majority of the weight, which is denoted as *retrieval head*  $h^*$ . As a result, the learned head weights are practically a head selector, a fact that can also be exploited to make test-time retrieval more efficient.

**End-to-end Retrieval with Attention** To perform retrieval over a corpus, we first generate key vectors  $K_d^{B+1,h^*}$  of retrieval head for all document tokens offline and index them with FAISS library (Johnson et al., 2021). For each query token, we issue its vector ( $Q_q^{B+1,h^*}$ ) to the index to retrieve top- $K'$  document tokens, which yields a filtered set of documents, each of which has at least one token retrieved by a query token. We then fetch all tokens of filtered documents, compute relevance scores following Eq. 1, and return top- $K$  documents with the highest scores  $r_{h^*}(q, d)$ . This is similar to the two-stage retrieval in ColBERT (Khattab and Zaharia, 2020), and we reuse their successful practice

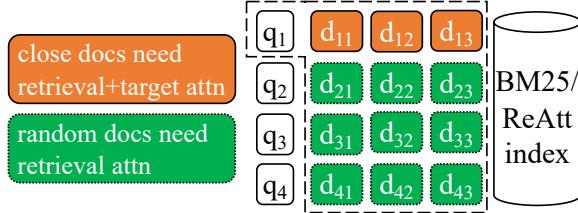


Figure 2: Illustration of approximate attention over the corpus with  $|Q|=4$  queries in a batch and  $K=3$  close documents per query. We use  $q_1$  as an example to illustrate the required computation, where **close documents** require both retrieval and target attention while **random documents** only require retrieval attention.

in index compression and search approximation to make test-time retrieval efficient, which we refer to [Santhanam et al. \(2021\)](#) for details.

### 2.3 How Good is Attention As-is?

To examine this question, we use T5-large and test queries from the Natural Question dataset (NQ), retrieve 100 documents with BM25, compute relevance scores  $r_h(\mathbf{q}, \mathbf{d})$  with half layers ( $B = 12$ ) as bi-encoder, and measure its correlation with the gold binary annotation. We found that among  $H = 24$  heads, 4 heads have non-trivial correlations of 0.137, 0.097, 0.082, and 0.059. We further perform end-to-end retrieval over Wikipedia using the best head, achieving top-10 retrieval accuracy of 43.5%, inferior to 55.5% of BM25. This demonstrates that there are indeed heads that can relate queries with relevant documents, but they are not competitive. We hypothesize that because self-attention is usually trained by comparing and relating tokens in a local context (512/1024 tokens) it cannot effectively identify relevant tokens in the enormous representation space of a corpus with millions of documents. This discrepancy motivates us to compute retrieval attention between queries and potentially all documents (i.e., attention over the corpus), and *adjust attention across documents to promote useful ones*.

## 3 Learning Retrieval as Attention

We first approximate attention over the corpus at training time by sub-sampling a manageable number of documents for each query containing both potentially relevant and random documents ([§ 3.1](#)). Next, we introduce our end-to-end training objective that optimizes a standard QA loss while also adding supervision to promote attention over documents that are useful for the end task ([§ 3.2](#)).

### 3.1 Approximate Attention over the Corpus

Encoding the entire corpus and computing attention between the query and all documents is very expensive. To make it practical, we propose to sub-sample a small set of documents for each query to approximate the whole corpus. Inspired by negative sampling methods used in dense retriever training ([Karpukhin et al., 2020; Xiong et al., 2021; Khattab and Zaharia, 2020](#)), we sub-sample both (1) documents close to queries that can be either relevant or hard negatives, and (2) random documents that are most likely to be easy negatives. This allows the model to distinguish between relevant and hard negative documents, while simultaneously preventing it from losing its ability to distinguish easy negatives, which form the majority of the corpus.

**Iterative Close Document Sub-sampling** To sample documents close to a query  $\mathcal{D}_q^{\text{close}}$ , we start from widely used lexical retriever BM25 ([Robertson and Zaragoza, 2009](#)) to retrieve  $K = 100$  documents, as shown by the orange blocks in [Fig. 2](#). We set  $K$  to a relatively large number to better approximate the local region, inspired by [Izacard and Grave \(2021b\)](#)'s findings that QA performance increases as more documents are used.

This fixed set of close documents can become outdated and no longer close to the query anymore as the retrieval attention gets better. To provide dynamic close sub-samples, we re-index the corpus and retrieve a new set of  $K$  documents using the current retrieval attention after each iteration. It is similar in spirit to the hard negative mining methods used in [Karpukhin et al. \(2020\); Khattab et al. \(2020\)](#), with a major difference that we do not manually or heuristically annotate documents but instead learn from the end loss with cross-document adjustment, which will be explained in [§ 3.2](#).

### In-batch Random Document Sub-sampling

We use close documents of other queries in the same batch as the random documents of the current query  $\mathcal{D}_q^{\text{random}} = \cup_{q' \in Q \wedge q' \neq q} \mathcal{D}_{q'}^{\text{close}}$  where  $Q$  contains all queries in a batch, as shown by the green blocks in [Fig. 2](#), which has the advantage of reusing document representations across queries. This is similar to the in-batch negatives used in DPR ([Karpukhin et al., 2020](#)) with a major difference that we reuse a token representations ( $K_d^{B+1,h}, 1 \leq h \leq H$ ) across queries instead of a single-vector document representation.

### 3.2 Cross-document Adjustment with Decoder-to-Encoder Attention Distillation

Given the sub-sampled  $|\mathcal{Q}| \times K$  documents  $\mathcal{D}_q = \mathcal{D}_q^{\text{close}} \cup \mathcal{D}_q^{\text{random}}$  for each query  $q$ , we compute the retrieval attention-based relevance scores  $r(q, d)$  and adjust them across multiple documents  $d \in \mathcal{D}_q$  only relying on end task supervision. Since retrieval is simply a means to achieve the downstream task, documents useful for downstream tasks should be promoted by retrieval. Inspired by reader-to-retriever distillation (Izacard and Grave, 2021a; Yang and Seo, 2020), we measure document usefulness based on cross-attention between decoder and encoder, and minimize retrieval attention’s discrepancy from it through distillation. In contrast to Izacard and Grave (2021a) that learns two models iteratively and alternatively, we optimize QA and distillation loss in a single model simultaneously.

**Minimizing KL-divergence Between Retrieval and Target Attention** Specifically, we denote cross-attention before softmax of the first position/token of the last decoder layer as *target attention*  $C_{a,q,\mathcal{D}_q} \in \mathbb{R}^{H \times |\mathcal{D}_q| \times (|d|+|q|)}$  where  $a$  is the answer,  $|\mathcal{D}_q|$  is the number of sub-sampled documents to be fused by the decoder (§ 2.1), and  $|d|$  is document length.<sup>2</sup> To aggregate token-level target attention into document-level distribution  $P^{\text{tgt}}(a, q, \mathcal{D}_q) \in \mathbb{R}^{|\mathcal{D}_q|}$ , we first perform softmax over all tokens in all query-document pairs ( $|\mathcal{D}_q| \times (|d|+|q|)$ ), sum over tokens of each query-document pair ( $|d|+|q|$ ), then average across multiple heads ( $H$ ):

$$P^{\text{tgt}}(a, q, \mathcal{D}_q) = \text{avg}_0 \left( \text{sum}_2 \left( \text{softmax}_{1,2}(C_{a,q,\mathcal{D}_q}) \right) \right).$$

Given relevance scores obtained from retrieval attention, the final cross-document adjustment loss is the KL-divergence between relevance distribution  $P^{\text{ret}}$  and target distribution  $P^{\text{tgt}}$ :

$$\begin{aligned} P^{\text{ret}}(q, \mathcal{D}_q) &= \text{softmax}(r(q, d_1), \dots, r(q, d_{|\mathcal{D}_q|})). \\ \mathcal{L}_{\text{cross-doc}} &= \text{KL}\left(\overline{P^{\text{tgt}}(a, q, \mathcal{D}_q)} \middle\| P^{\text{ret}}(q, \mathcal{D}_q)\right), \end{aligned} \quad (2)$$

where the overline indicates stop gradient back propagation to target distributions. Our final loss combines QA loss and cross-document adjustment loss with  $\alpha$  as combination weight.

$$\mathcal{L} = \mathcal{L}_{\text{QA}} + \alpha \cdot \mathcal{L}_{\text{cross-doc}}. \quad (3)$$

<sup>2</sup>We also attempted other variations of target attention and found performances are similar, consistent with observations in Izacard and Grave (2021a).

**Zero Target Attention for Random Documents** For a batch with  $|\mathcal{Q}|$  queries, we need to compute retrieval attention and target attention between  $|\mathcal{Q}| \times |\mathcal{Q}| \times K$  query-document pairs. This is both computation- and memory-intensive when batch size is large, especially for target attention because it requires  $L - B$  layers of joint encoding of query-document pairs in the cross-encoder. To alleviate this, we make a simple and effective assumption that in-batch random documents are not relevant to the current query thus having zero target attention:  $P^{\text{tgt}}(a, q, \mathcal{D}_q^{\text{random}}) \in \mathbb{R}^{|\mathcal{D}_q^{\text{random}}|} \leftarrow 0$ . As a result, we only need to run cross-encoder and decoder for  $K$  close documents of each query, as shown in Fig. 2. In Appendix A we will introduce our efficient implementation to make it possible to run a large batch size over a limited number of GPUs.

### 3.3 Domain Adaptation Methods

One of the major benefits of a single end-to-end trainable model is that given a new corpus from a new domain, possibly without retrieval annotations, we can easily adapt it by end-to-end training. This section describes how we adapt ReAtt under different setups.

We consider adapting ReAtt with (1) QA supervision, (2) information retrieval (IR) supervision, or (3) unsupervised adaptation where we only have access to the document corpus. Although our goal is to learn retrieval through downstream tasks instead of retrieval supervision, being able to consume retrieval annotations is helpful when retrieval supervision is indeed available. To do so, we convert retrieval task with annotations in the form of query-document-relevance triples  $\langle q, d, l \rangle$  into a generative task: given a query, the target is to generate *its relevant document and the corresponding relevance* with the following format “relevance:  $l$ .  $d$ ”. If a query has multiple relevant documents, we follow Izacard and Grave (2021b) to randomly sample one of them. For unsupervised adaptation, with simplicity as our primary goal, we randomly choose one sentence from a document and mask one entity, which is considered as the “query”, and have our model generate the masked entity as the “answer”, similar to salient span masking (SSM) used in Guu et al. (2020).

## 4 In-domain Experiments

In this section, we examine if supervisedly training ReAtt end-to-end with *only* QA supervision yields both competitive retrieval and QA performance.

**Datasets, Baselines, and Metrics** We train our model using the Natural Questions dataset (NQ). We compare retrieval performance with lexical models BM25 (Robertson and Zaragoza, 2009), passage-level dense retrievers DPR, ANCE, coCondenser, FiD-KD, YONO (with and without retrieval pretraining) (Karpukhin et al., 2020; Oguz et al., 2021; Xiong et al., 2021; Gao and Callan, 2022; Izacard and Grave, 2021a; Lee et al., 2021a), and token/phrase-level dense retrievers DensePhrase, ColBERT, ColBERT-NQ (Lee et al., 2021b; Khattab and Zaharia, 2020; Khattab et al., 2020).<sup>3</sup> Among them ColBERT-NQ, FiD-KD and YONO are the most fair-to-compare baselines because of either similar token-level retrieval granularity (ColBERT-NQ) or similar end-to-end training settings (FiD-KD and YONO). We report top-k retrieval accuracy (R@k), the fraction of queries with at least one retrieved document containing answers. We compare QA performance with ORQA, REALM, RAG, FiD, EMDR<sup>2</sup>, YONO, UnitedQA, and R2-D2 (Lee et al., 2019; Guu et al., 2020; Lewis et al., 2020; Izacard and Grave, 2021b,a; Sachan et al., 2021; Lee et al., 2021a; Cheng et al., 2021; Fajcik et al., 2021) using exact match (EM), among which FiD, EMDR<sup>2</sup>, and YONO are the most fair-to-compare baselines because they have similar model sizes and training settings.

## 5 Implementation Details of ReAtt

ReAtt is based on T5-large with  $B = 12$  encoder layers as bi-encoder and temperatures  $\tau = 0.001$  to select the best retrieval head. We retrieve  $K = 100$  close documents for each query, and use a batch size of  $|\mathcal{Q}| = 64$  queries to obtain in-batch random documents. We use  $\alpha = 8$  to combine cross-document adjustment loss with QA loss. We use AdamW with a learning rate of 5e-5, 10% steps of warmup, and linear decay. We first warmup cross-attention’s ability to distinguish documents by only using the QA loss for 3K steps, then train with the combined losses (Eq. 3) for 4 iterations, where the first iteration uses close documents returned by BM25, and the following 3 iterations use close documents returned by the previous ReAtt model (denoted as ReAtt<sub>BM25</sub>). Each iteration has 8K update steps and takes  $\sim 1.5$  days on a single node with  $8 \times$  A100 GPUs with 80GB memory. Since DPR (Karpukhin et al., 2020) achieves stronger performance than BM25, training with close doc-

Models	R@1	R@5	R@20	R@100	#Params.
<i>supervised retrievers</i>					
BM25	23.9	45.9	63.8	78.9	-
DPR	45.9	68.1	80.0	85.9	220M
DPR <sup>new</sup>	52.5	72.2	81.3	87.3	220M
DPR-PAQ	-	74.2	84.0	89.2	220M
ANCE	-	-	81.9	87.5	220M
coCondenser	-	75.8	84.3	89.0	220M
DensePhrase	51.1	69.9	78.7	-	330M
ColBERT	-	-	79.1	-	110M
ColBERT-NQ	54.3	75.7	85.6	90.0	110M
<i>semi/unsupervised retrievers</i>					
FiD-KD	49.4	73.8	84.3	89.3	220M
YONO <sub>w/o PT</sub>	-	-	72.3	82.2	165M
YONO <sub>w/ PT</sub>	-	75.3	85.2	90.2	165M
ReAtt DPR	54.6	77.2	<b>86.1</b>	<b>90.7</b>	165M
ReAtt BM25	<b>55.8</b>	<b>77.4</b>	86.0	90.4	165M

Table 1: Retrieval performance on NQ. PT is retrieval pretraining. Fair-to-compare baselines are highlighted with background color. Best performance is in bold.

uments returned by DPR can potentially reduce training time. We experimented with training on close documents from DPR for a single iteration with 16K steps (denoted as ReAtt<sub>DPR</sub>). Since both approaches achieve similar performance (Tab. 1 and Tab. 2) and ReAtt<sub>DPR</sub> is cheaper to train, we use it in other experimental settings.

At test-time, we save key vectors of all tokens in the corpus and use exact index from FAISS (i.e., `faiss.IndexFlatIP`) to perform inner-product search. We retrieve  $K' = 2048$  document tokens for each query token and return top-100 documents with the highest aggregated scores (Eq. 1) to generate answers. We found compressing index with clustering and quantization proposed by Santhanam et al. (2021) can greatly reduce search latency and index size with a minor retrieval accuracy loss.

## 5.1 Overall Results

We compare ReAtt with various retrievers and readers in Tab. 1 and Tab. 2. ReAtt achieves both slightly better retrieval performance than the strongest retriever baseline ColBERT-NQ (Khattab et al., 2020) and comparable QA performance than the strong reader baseline FiD-KD (Izacard and Grave, 2021a) on NQ, demonstrating for the first time that fully end-to-end training using QA supervision can produce both competitive retrieval and QA performance. Compared to another single-model architecture YONO (Lee et al., 2021a), ReAtt offers better performance without cumbersome pretraining to warm-up retrieval.

<sup>3</sup>ColBERT is trained on MS MARCO, ColBERT-NQ is on NQ.

Models	EM	#Params.
ORQA (Lee et al., 2019)	33.3	330M
REALM (Guu et al., 2020)	40.4	330M
RAG (Lewis et al., 2020)	44.5	220M
FiD (Izacard and Grave, 2021b)	51.4	990M
FiD-KD (Izacard and Grave, 2021a)	54.4	990M
EMDR <sup>2</sup> (Sachan et al., 2021)	52.5	440M
YONO <sub>w/o</sub> PT (Lee et al., 2021a)	42.4	440M
YONO <sub>w/</sub> PT (Lee et al., 2021a)	53.2	440M
UnitedQA (Cheng et al., 2021)	54.7	1.870B
R2-D2 (Fajcik et al., 2021)	<b>55.9</b>	1.290B
ReAtt DPR	54.0	770M
ReAtt BM25	54.7	770M

Table 2: QA performance on NQ. PT is retrieval pre-training. Fair-to-compare baselines are highlighted. Best performance is in bold.

## 5.2 Ablations

We perform ablation experiments to understand the contribution of each component. Due to resource limitations, all ablations are trained with 2K steps per iteration. We use ReAtt trained with  $B=12$  bi-encoder layers,  $|Q|=16$  batch size, and  $\alpha=8$  cross-document loss weight as the baseline, remove one component or modify one hyperparameter at a time to investigate its effect. As shown in Tab. 3, we found: 1. Only using QA loss without cross-document adjustment (#2) improves retrieval performance over the original T5 (#3), but cross-document adjustment is necessary to achieve further improvement (#1). 2. Iteratively retrieving close documents with the current model is helpful (#5 vs #1). 3. In-batch random documents are beneficial (#4 vs #1), and a larger batch size leads to larger improvements (#8-11). 4. A larger weight on cross-document adjustment loss improves retrieval performance but hurts QA performance, with  $4 \sim 8$  achieving a good trade-off (#12-15). 5. A small number of bi-encoder layers (#6) significantly hurts retrieval while a large number of layers (#7) significantly hurts QA, suggesting choosing equal numbers of layers in bi-encoder and cross-encoder.

## 6 Out-of-domain Generalization and Adaptation

In this section, we examine both zero-shot retrieval performance on out-of-domain datasets and ReAtt’s end-to-end adaptability in supervised (QA, IR) and unsupervised settings.

### 6.1 Datasets, Baselines, and Metrics

We choose 7 datasets from BEIR (Thakur et al., 2021), a benchmark covering diverse domains

#	Methods	R@1	R@5	R@20	R@100	EM
<i>ReAtt baseline with <math>B=12</math>, <math> Q =16</math>, <math>\alpha=8</math></i>						
1		41.9	68.8	82.5	88.9	46.3
<i>remove one component</i>						
2	- cross-doc loss	21.7	49.0	71.5	83.5	46.0
3	- QA (=T5)	13.2	33.7	53.6	67.7	3.0
4	- in-batch	38.1	66.0	80.3	87.6	46.7
5	- iterative	41.2	68.3	82.0	88.4	45.0
<i>different #layers in bi-encoder <math>B</math></i>						
6	$B=6$	19.1	42.1	62.4	78.1	40.3
7	$B=18$	38.2	63.8	79.3	87.4	35.2
<i>different batch sizes <math>Q</math></i>						
8	$ Q =4$	39.4	66.1	80.7	88.1	45.0
9	$ Q =8$	40.7	67.1	82.1	88.6	45.7
10	$ Q =32$	43.6	69.4	82.8	89.1	46.4
11	$ Q =64$	45.5	71.0	83.3	89.4	47.3
<i>different cross-doc loss weights <math>\alpha</math></i>						
12	$\alpha=1$	37.4	65.4	80.9	88.0	47.3
13	$\alpha=2$	39.7	66.9	81.7	88.4	47.4
14	$\alpha=4$	40.9	68.0	82.1	88.8	46.9
15	$\alpha=16$	42.0	68.8	82.5	88.8	45.5

Table 3: Ablations by removing one component or changing one hyperparameter from the ReAtt baseline.

and tasks. On each dataset we compare ReAtt with different types of retrievers including BM25, DPR, and ColBERT. We consider 2 QA datasets (BioASQ and FiQA (Tsatsaronis et al., 2015; Maia et al., 2018)) and one IR dataset (MS MARCO (Nguyen et al., 2016)) to evaluate supervised adaptation capability, and 4 other datasets (CQA DupStack, TREC-COVID, SCIDOCs, SciFact (Hoogeveen et al., 2015; Voorhees et al., 2020; Cohan et al., 2020; Wadden et al., 2020)) to evaluate unsupervised adaptation capability. Detailed statistics are listed in Tab. 8. We report nDCG@10 to measure retrieval performance and EM to measure QA performance. We group all baselines into three categories and denote them with different colors in the following tables:

- **Supervised adaptation models** are trained with downstream task supervision, including RAG trained on BioASQ, Contriever fine-tuned on FiQA, and docT5query, ANCE, ColBERT, and Contriever fine-tuned on MS MARCO (Nogueira and Lin, 2019; Xiong et al., 2021; Khattab and Zaharia, 2020; Izacard et al., 2021).
- **Unsupervised adaptation models** are trained on domain corpus in an unsupervised way such as contrastive learning or pseudo query generation, including SimCSE and TSDAE+GPL (Gao et al., 2021c; Wang et al., 2021a,b).
- **Pretraining models** are trained on corpora with-

Tasks Datasets	QA		Retrieval	
	BioASQ	FiQA	MS MARCO	
<i>zero-shot performance</i>				
BM25	68.1	23.6		22.8
DPR	14.1	11.2		17.7
ColBERT-NQ	65.5	23.8	<b>32.8</b>	
ReAtt	<b>71.1</b>	<b>30.1</b>		32.3
<i>additional training</i>				
Contriever	-	32.9	docT5query	33.8
SimCSE	58.1	31.4	ANCE	38.8
TSDAE+GPL	61.6	34.4	ColBERT	40.1
Contriever <sub>w/ FT</sub>	-	38.1	Contriever	<b>40.7</b>
ReAtt	+5.8 <b>76.9</b>	+8.5 <b>38.6</b>	ReAtt	+7.6 39.9

Table 4: nDCG@10 of zero-shot and supervised adaptation experiments on two QA and one IR datasets. We use colors to denote categories: **pretraining**, **unsupervised adaptation**, and **supervised adaptation**. Baselines comparable to ReAtt are highlighted with blue background color. We also show the improvement of ReAtt over zero-shot performance in subscript.

# Ablations	nDCG@1 @5		EM
1 RAG	14.6	13.0	1.3
2 + reader	14.6	13.0	-
3 + qry enc (e2e)	0.0	0.0	-13.0
4 + doc/qry enc*	29.4	27.1	14.1
5 + reader (pipe)	29.4	27.1	-
6 + qry enc	23.3	23.2	-4.0
7 T5	49.2	47.7	0.0
8 + e2e	75.2	73.5	25.7
9 ReAtt	72.8	70.1	17.2
10 + e2e	<b>77.4</b>	<b>75.4</b>	<b>5.3</b>
			<b>47.2</b> <b>30.0</b>

Table 5: RAG and ReAtt on BioASQ. Each indent indicates fine-tuning one more component than its parent with performance difference colored with green/red. \* denotes fine-tuning conducted sequentially instead of jointly with the current component.

out direct exposure to the target domain, such as Contriever (Izacard et al., 2021) trained with contrastive learning on Wikipedia and CCNet. We highlight baselines in the same category as ReAtt in the following tables since comparison between them is relatively fair. Details of adaptation of ReAtt can be found in Appendix B.

## 6.2 Experimental Results

Results of supervised and unsupervised adaptation are listed in Tab. 4, Tab. 5, and Tab. 6 respectively.

**Zero-shot Generalization Ability** As shown in Tab. 4 and Tab. 6, the zero-shot performance of ReAtt is significantly better than other zero-shot baselines on two QA datasets and one fact checking dataset (+3.0/+6.5/+4.5 on BioASQ/FiQA/SciFact than the second best), and overall comparable on the rest of datasets (-0.5/-0.6/-3.0/-1.0 on MS

Methods	CQA.	TRECC.	SCIDOCs	SciFact
<i>zero-shot performance</i>				
BM25	29.9	<b>65.6</b>	<b>15.8</b>	66.5
DPR	15.3	33.2	7.7	31.8
ANCE	29.6	65.4	12.2	50.7
ColBERT-NQ	<b>33.9</b>	48.9	15.6	65.3
ReAtt	33.3	62.6	14.8	<b>71.0</b>
<i>additional training</i>				
Contriever	34.5	59.6	<b>16.5</b>	67.7
SimCSE	29.0	68.3	-	55.0
TSDAE+GPL	35.1	74.6	-	68.9
ReAtt	+3.3 <b>36.6</b>	+13.4 <b>76.0</b>	+1.0 15.8	+0.2 <b>71.2</b>

Table 6: nDCG@10 of zero-shot and unsupervised adaptation on four datasets. Format is similar to Tab. 4

MARCO/CQA./TRECC./SCIDOCs than the best which is usually BM25), demonstrating that our end-to-end training with QA loss on NQ produces a robust retriever. We conjecture that the superior performance on QA datasets can be attributed to our end-to-end training using QA loss which learns retrieval that better aligns with the end task than training with retrieval annotations.

## Retrieval Adaptation with QA Supervision

As shown in the left-hand side of Tab. 4, end-to-end adaptation with QA supervision significantly improves ReAtt’s retrieval performance by 5.8/8.5 on BioASQ/FiQA, achieving similar performance as Contriever fine-tuned on FiQA, and better performance than other unsupervised methods, confirming the end-to-end adaptability of our methods.

**End-to-end QA Adaptation** We perform end-to-end adaptation on BioASQ and compare with RAG as a baseline, which combines DPR as retriever and BART as reader, and DPR has a query and document encoder. Since updating document encoder requires corpus re-indexing, it is fixed during fine-tuning. We found end-to-end fine-tuning fails on RAG. To understand why, we conduct a rigorous experiment that breaks down each component of RAG to find the failure point in Tab. 5.

Starting from the initial model trained on NQ (#1), we first fine-tune the reader while fixing the query encoder (#2), and as expected QA performance improves. However fine-tuning both query encoder and reader (end-to-end #3) makes the retriever collapse with zero relevant documents returned, indicating end-to-end fine-tuning does not work for RAG on new domains. In order to improve both retrieval and QA, we need to fine-tune RAG in a pipeline manner: first fine-tune the re-

triever (both query and doc encoder) similarly to DPR using retrieval annotations (#4), then fine-tune the reader (#5). With the DPR-like fine-tuned retriever, end-to-end fine-tuning of query encoder and reader still fails (#6), although the retriever does not completely collapse.

End-to-end fine-tuning of ReAtt improves retrieval and QA simultaneously. Fine-tuning starting from ReAtt trained on NQ is better than starting from T5, indicating the capability learned in NQ could be transferred to BioASQ. Comparing RAG and ReAtt, we identify several keys that enable end-to-end adaptation. (1) ReAtt relying on token-level attention has a strong initial performance, (2) cross-document adjustment over both close and random documents in ReAtt provides a better gradient estimation than only using retrieved documents in RAG, (3) distillation-based loss in ReAtt might be more effective than multiplying the retrieval probability into the final generation probability.

**Leveraging Retrieval Annotations** As shown on the right-hand side of Tab. 4, ReAtt is able to consume retrieval supervision in a generative format and achieve competitive performance as other supervised dense retrievers.

**Unsupervised Adaptation with SSM** As shown in Tab. 6, adaptation by simply masking salient entities from sentences as input and generating masked entities using ReAtt improves the retrieval performance on 4 datasets, some by a large margin, achieving comparable or superior performance than strong retrieval adaptation methods such as TSDAE+GPL that relies on query generation. This indicates that our end-to-end trainable model also works well in unsupervised settings without involving too many engineering heuristics.

## 7 Related Work

Retrieval-augmented question answering utilizes evidence retrieved from an external knowledge source to facilitate question answering. There have been several attempts to learn retrievers and readers jointly. ORQA, REALM, RAG, EMDR<sup>2</sup>, YONO, and Atlas (Lee et al., 2019; Guu et al., 2020; Sachan et al., 2021; Lee et al., 2021a; Izacard et al., 2022) first warm-up retrievers using unsupervised pre-training methods such as inverse cloze task (ICT), salient span masking (SSM), and large-scale contrastive learning, or initialize from supervised retrievers, then fine-tune both retriever and reader on

downstream tasks. They either use fixed index (Lee et al., 2019; Lewis et al., 2020) or asynchronously update the index during training (Guu et al., 2020; Sachan et al., 2021; Lee et al., 2021a; Izacard et al., 2022). Recently, retrieval-augmented models are scaled up to very large corpora such as the web (Piktus et al., 2021; Borgeaud et al., 2021), making them capable of handling information out of the scope of Wikipedia. Atlas (Izacard et al., 2022) scales up retrieval-augmented models with T5-11B as the reader and Contriever (Izacard et al., 2021) as the retriever and achieves strong few-shot performance on multiple benchmarks. Detailed comparisons of these models can be found in Tab. 7. More related works on dense retrieval, unsupervised retrieval learning, and retrieval augmentation for language modeling can be found in Appendix C.

## 8 Conclusion

We propose retrieval as attention (ReAtt), a single Transformer model that can be learned in an end-to-end fashion only using end task loss. We demonstrated on NQ dataset that ReAtt can achieve both competitive retrieval and QA performance. We further show that ReAtt is easy to adapt to other domains in both supervised and unsupervised settings, achieving both boosted retrieval and end task performance. Future directions include better end-to-end training objectives and efficient training and inference.

## Limitations

ReAtt is based on token-level representations, and belongs to the same category as token-level dense retrievers such as ColBERT (Khattab and Zaharia, 2020). Comparing to passage-level dense retrievers such as DPR (Karpukhin et al., 2020), token-level retrievers usually offer better performance (shown in Tab. 1, Tab. 4, and Tab. 6) but require more space to store the index and longer query time. Our methods have the same limitation. We found ColBERT’s practice in index compression and approximate search (Khattab and Zaharia, 2020; Santhanam et al., 2021, 2022) also works for our model, making this issue less of a concern.

## Acknowledgments

This work was supported by a gift from Bosch Research. We would like to thank Chunting Zhou, Uri Alon, Omar Khattab, Patrick Lewis, and Jane Dwivedi-Yu for their insightful feedback and help with experiments.

## References

- Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2021. *Improving language models by retrieving from trillions of tokens*. *CoRR*, abs/2112.04426.
- Hao Cheng, Yelong Shen, Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2021. *Unit-edqa: A hybrid approach for open domain question answering*. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 3080–3090. Association for Computational Linguistics.
- Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel S. Weld. 2020. *SPECTER: document-level representation learning using citation-informed transformers*. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2270–2282. Association for Computational Linguistics.
- Martin Fajcik, Martin Docekal, Karel Ondrej, and Pavel Smrz. 2021. *R2-D2: A modular baseline for open-domain question answering*. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pages 854–870. Association for Computational Linguistics.
- Luyu Gao and Jamie Callan. 2021. *Condenser: a pre-training architecture for dense retrieval*. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 981–993. Association for Computational Linguistics.
- Luyu Gao and Jamie Callan. 2022. *Unsupervised corpus aware language model pre-training for dense passage retrieval*. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 2843–2853. Association for Computational Linguistics.
- Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021a. *COIL: revisit exact lexical match in information retrieval with contextualized inverted list*. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 3030–3042. Association for Computational Linguistics.
- Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. 2021b. *Scaling deep contrastive learning batch size under memory limited setup*. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*, pages 316–321, Online. Association for Computational Linguistics.
- Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021c. *Simcse: Simple contrastive learning of sentence embeddings*. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 6894–6910. Association for Computational Linguistics.
- Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. *REALM: retrieval-augmented language model pre-training*. *CoRR*, abs/2002.08909.
- Doris Hoogeveen, Karin M. Verspoor, and Timothy Baldwin. 2015. *Cquadupstack: A benchmark data set for community question-answering research*. In *Proceedings of the 20th Australasian Document Computing Symposium, ADCS 2015, Parramatta, NSW, Australia, December 8-9, 2015*, pages 3:1–3:8. ACM.
- Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. *Towards unsupervised dense information retrieval with contrastive learning*. *CoRR*, abs/2112.09118.
- Gautier Izacard and Edouard Grave. 2021a. *Distilling knowledge from reader to retriever for question answering*. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.
- Gautier Izacard and Edouard Grave. 2021b. *Leveraging passage retrieval with generative models for open domain question answering*. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 874–880. Association for Computational Linguistics.
- Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. *Few-shot learning with retrieval augmented language models*. *CoRR*, abs/2208.03299.
- Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. *Billion-scale similarity search with gpus*. *IEEE Trans. Big Data*, 7(3):535–547.
- Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. *Dense passage retrieval for open-domain question answering*. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020*,

*Online*, November 16-20, 2020, pages 6769–6781. Association for Computational Linguistics.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. **Generalization through memorization: Nearest neighbor language models**. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Omar Khattab, Christopher Potts, and Matei Zaharia. 2020. **Relevance-guided supervision for openqa with colbert**. *CoRR*, abs/2007.00814.

Omar Khattab and Matei Zaharia. 2020. **Colbert: Efficient and effective passage search via contextualized late interaction over BERT**. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 39–48. ACM.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. **Natural questions: a benchmark for question answering research**. *Trans. Assoc. Comput. Linguistics*, 7:452–466.

Haejun Lee, Akhil Kedia, Jongwon Lee, Ashwin Paranjape, Christopher D. Manning, and Kyoung-Gu Woo. 2021a. **You only need one model for open-domain question answering**. *CoRR*, abs/2112.07381.

Jinhyuk Lee, Alexander Wettig, and Danqi Chen. 2021b. **Phrase retrieval learns passage retrieval, too**. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 3661–3672. Association for Computational Linguistics.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. **Latent retrieval for weakly supervised open domain question answering**. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 6086–6096. Association for Computational Linguistics.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. **Retrieval-augmented generation for knowledge-intensive NLP tasks**. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Ji Ma, Ivan Korotkov, Yinfei Yang, Keith B. Hall, and Ryan T. McDonald. 2021. **Zero-shot neural passage retrieval via domain-targeted synthetic question generation**. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 1075–1088. Association for Computational Linguistics.

Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. 2018. **Www’18 open challenge: Financial opinion mining and question answering**. In *Companion of the The Web Conference 2018 on The Web Conference 2018, WWW 2018, Lyon , France, April 23-27, 2018*, pages 1941–1942. ACM.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. **MS MARCO: A human generated machine reading comprehension dataset**. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016*, volume 1773 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Rodrigo Nogueira and Jimmy Lin. 2019. **From doc2query to docttttquery**.

Barlas Oguz, Kushal Lakhotia, Anchit Gupta, Patrick S. H. Lewis, Vladimir Karpukhin, Aleksandra Piktus, Xilun Chen, Sebastian Riedel, Wen-tau Yih, Sonal Gupta, and Yashar Mehdad. 2021. **Domain-matched pre-training tasks for dense retrieval**. *CoRR*, abs/2107.13602.

Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Dmytro Okhonko, Samuel Broscheit, Gautier Izacard, Patrick Lewis, Barlas Oguz, Edouard Grave, Wen-tau Yih, and Sebastian Riedel. 2021. **The web is your oyster - knowledge-intensive NLP against a very large web corpus**. *CoRR*, abs/2112.09924.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. **Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering**. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 5835–5847. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. **Exploring the limits of transfer learning with a unified text-to-text transformer**. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Stephen E. Robertson and Hugo Zaragoza. 2009. **The probabilistic relevance framework: BM25 and beyond**. *Found. Trends Inf. Retr.*, 3(4):333–389.

- Devendra Singh Sachan, Siva Reddy, William L. Hamilton, Chris Dyer, and Dani Yogatama. 2021. **End-to-end training of multi-document reader and retriever for open-domain question answering**. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 25968–25981.
- Keshav Santhanam, Omar Khattab, Christopher Potts, and Matei Zaharia. 2022. **PLAID: an efficient engine for late interaction retrieval**. *CoRR*, abs/2205.09707.
- Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2021. **Colbertv2: Effective and efficient retrieval via lightweight late interaction**. *CoRR*, abs/2112.01488.
- Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. **BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models**. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks I, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.
- George Tsatsaronis, Georgios Balikas, Prodromos Malakasiotis, Ioannis Partalas, Matthias Zschunke, Michael R. Alvers, Dirk Weissenborn, Anastasia Krithara, Sergios Petridis, Dimitris Polychronopoulos, Yannis Almirantis, John Pavlopoulos, Nicolas Baskiotis, Patrick Gallinari, Thierry Artières, Axel-Cyrille Ngonga Ngomo, Norman Heino, Éric Gaussier, Liliana Barrio-Alvers, Michael Schroeder, Ion Androutsopoulos, and Georgios Paliouras. 2015. **An overview of the BIOASQ large-scale biomedical semantic indexing and question answering competition**. *BMC Bioinform.*, 16:138:1–138:28.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. **Attention is all you need**. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.
- Ellen M. Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R. Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. 2020. **TREC-COVID: constructing a pandemic information retrieval test collection**. *SIGIR Forum*, 54(1):1:1–1:12.
- David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. 2020. **Fact or fiction: Verifying scientific claims**. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 7534–7550. Association for Computational Linguistics.
- Kexin Wang, Nils Reimers, and Iryna Gurevych. 2021a. **TSDAE: using transformer-based sequential denoising auto-encoder for unsupervised sentence embedding learning**. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pages 671–688. Association for Computational Linguistics.
- Kexin Wang, Nandan Thakur, Nils Reimers, and Iryna Gurevych. 2021b. **GPL: generative pseudo labeling for unsupervised domain adaptation of dense retrieval**. *CoRR*, abs/2112.07577.
- Yuhuai Wu, Markus Norman Rabe, DeLesley Hutchins, and Christian Szegedy. 2022. **Memorizing transformers**. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.
- Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. **Approximate nearest neighbor negative contrastive learning for dense text retrieval**. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.
- Sohee Yang and Minjoon Seo. 2020. **Is retriever merely an approximator of reader?** *CoRR*, abs/2010.10999.
- Zexuan Zhong, Tao Lei, and Danqi Chen. 2022. **Training language models with memory augmentation**. *CoRR*, abs/2205.12674.

Model	Architecture			Retriever training			Granu.
	Retriever	Reader	Single	Init.	Warm-up	End-to-end loss	
ORQA (Lee et al., 2019)	BERT	BERT	✗	BERT	ICT	Prob. marginalization	Passage
REALM (Guu et al., 2020)	BERT	BERT	✗	BERT	ICT, SSM	Prob. marginalization	Passage
RAG (Lewis et al., 2020)	BERT	BART	✗	DPR	-	Prob. marginalization	Passage
EMDR <sup>2</sup> (Sachan et al., 2021)	BERT	T5	✗	BERT	ICT, SSM	Expectation maximization	Passage
YONO (Lee et al., 2021a)	T5	T5	✓	T5	SSM	Attention distillation	Passage
Atlas (Izacard et al., 2022)	BERT	T5	✗	Contriever	MLM	Perplexity distillation	Passage
ReAtt	T5	T5	✓	T5	-	Attention distillation	Token

Table 7: Detailed comparison between end-to-end retriever-reader models. ICT is inverse cloze task, SSM is salient span masking, and MLM is masked language modeling. Granu. is retrieval granularity.

Dataset	Domain	Task	Train		Test	
			#Queries	#Annotations	#Queries	Corpus
<i>In-domain</i>						
NQ	Wiki	Question answering	79K	133K	3,610	21.015M
<i>Out-of-domain supervised adaptation</i>						
BioASQ	Biomed	Question answering	3K	32K	500	1.000M
FiQA	Finance	Question answering	6K	14K	648	58K
MS MARCO	Misc	Information retrieval	503K	533K	6,980	8.842M
<i>Out-of-domain unsupervised adaptation</i>						
CQADupStack	StackExchange	Duplicate question retrieval	-	-	13,145	457K
TREC-COVID	Biomed	Information retrieval	-	-	50	171K
SCIDOCs	Science	Citation prediction	-	-	1,000	26K
SciFact	Science	Fact checking	1K	1K	300	5K

Table 8: Statistics of 8 datasets categorized by experimental settings, including the number of training/test queries, retrieval annotations (query-document pairs), and documents in the corpus.

## A Efficient Implementation

Under typical optimization setups where the loss is point-wise with respect to each training data, like training classifiers or readers, scaling batch size can be easily achieved with gradient accumulation. However, due to the use of in-batch negatives, our systems, like others (Karpukhin et al., 2020; Qu et al., 2021), require having all examples in a batch to reside in GPUs simultaneously when trained directly. Larger batches therefore need proportionally more GPU memory.

In order to accommodate large batches with our limited memory hardware, we adopt the gradient cache approach (Gao et al., 2021b) decouple instances in the same batch. In particular, we run an extra forward pass over the large batch in inference mode and record (1) representations for all query and document tokens ( $Q_q^{B+1,h}$  and  $K_d^{B+1,h}$ ) and (2) decoder-encoder target attention values ( $C_{a,q,D_q}$ ). Note that we *do not* store model internal activation nor perform gradient computation with respect to model parameters in this step. With (1) we can compute the retrieval attention, and with (2) we can compute cross-document adjustment loss (Eq. 2). We then compute and cache gradient vec-

tors of all query and document vectors with respect to Eq. 2. We finally optimize the model with a sufficiently small batch size to fit in GPU memory and use cached gradient in the backward pass of the Eq. 2.

## B Details of Adaptation Experiments

For supervised adaptation, we train on BioASQ, FiQA, and MS MARCO separately using all training queries. For CQADupStack, we merge the document corpora of 12 sub-domains into a single corpus to sample masked sentences for salient span masking training. For each of the 4 unsupervised domain adaptation datasets (CQADupStack, TREC-COVID, SCIDOCs, SciFact), we sample 20~100K sentences and mask one entity, which is approximately proportional to the size of the corpus with a larger sampling rate for small corpora. We reuse the same hyperparameters as NQ (§ 5), except that we train each model for a single iteration using close documents from BM25 with 4K update steps and a batch size of 16. Since MS MARCO has a large number of annotations, we train for 12K update steps.

## C Related Work

**Dense Retrieval Models** Dense retrieval models can be categorized into two groups, passage-level retrievers (Karpukhin et al., 2020; Oguz et al., 2021; Xiong et al., 2021; Gao and Callan, 2022) and token/phrase-level retrievers (Khattab and Zaria, 2020; Khattab et al., 2020; Gao et al., 2021a; Lee et al., 2021b). Passage-level retrievers encode queries and documents into a single vector, while token/phrase-level retrievers directly use token/phrase representations, resulting in multi-vector representations. Passage-level retrievers are usually more efficient but less expressive than token-level retrievers.

**Unsupervised Retrieval Learning** Unsupervised retrieval learning methods can be categorized into two types: pretraining-based (Lee et al., 2019; Gao et al., 2021c; Wang et al., 2021a; Gao and Callan, 2021; Izacard et al., 2021) and question generation-based (Ma et al., 2021; Wang et al., 2021b). SimCSE (Gao et al., 2021c) obtains representations of the same input by passing through the model twice with different dropout masks and minimizes their distance. Contriever (Izacard et al., 2021) is trained by large-scale contrastive learning with random cropping of text spans sampled from Wikipedia and CCNet. GPL (Wang et al., 2021b) leverages query generators to obtain pseudo queries, and collect positive and negative documents by pseudo labeling using a cross-encoder.

**Retrieval Augmentation for Language Modeling** Retrieval from external datastore to improve language modeling perplexity has been explored by many works, where additional tokens are retrieved during generation based on contextual representations (Khandelwal et al., 2020; Borgeaud et al., 2021; Wu et al., 2022; Zhong et al., 2022). They differ in whether retrieval is fixed or learnable, retrieval frequency, and contextual representations used to perform nearest neighbors search.

# Shall We Pretrain Autoregressive Language Models with Retrieval? A Comprehensive Study

Boxin Wang<sup>\*†‡</sup>Wei Ping<sup>\*†‡</sup>Peng Xu<sup>\*</sup>Lawrence McAfee<sup>2</sup>Zihan Liu<sup>2</sup>Mohammad Shoeybi<sup>2</sup>Yi Dong<sup>2</sup>Oleksii Kuchaiev<sup>2</sup>Bo Li<sup>1</sup>Chaowei Xiao<sup>2,3</sup>Anima Anandkumar<sup>2</sup>Bryan Catanzaro<sup>2</sup>

## Abstract

Large decoder-only language models (LMs) can be largely improved in terms of perplexity by retrieval (*e.g.*, RETRO), but its impact on text generation quality and downstream task accuracy is unclear. Thus, it is still an open question: *shall we pretrain large autoregressive LMs with retrieval?* To answer it, we perform a comprehensive study on a *scalable pretrained* retrieval-augmented LM (*i.e.*, RETRO) compared with standard GPT and retrieval-augmented GPT incorporated at fine-tuning or inference stages. We first provide the recipe to reproduce RETRO up to 9.5B parameters while retrieving a text corpus with 330B tokens. Based on that, we have the following novel findings: *i)* RETRO outperforms GPT on text generation with much less degeneration (*i.e.*, repetition), moderately higher factual accuracy, and slightly lower toxicity with a nontoxic retrieval database. *ii)* On the LM Evaluation Harness benchmark, RETRO largely outperforms GPT on knowledge-intensive tasks, but is on par with GPT on other tasks. Furthermore, we introduce a simple variant of the model, RETRO++, which largely improves open-domain QA results of original RETRO (*e.g.*, EM score +8.6 on Natural Question) and significantly outperforms retrieval-augmented GPT across different model sizes. Our findings highlight the promising direction of pretraining autoregressive LMs with retrieval as future foundation models. We release our implementation at: <https://github.com/NVIDIA/Megatron-LM#retro>.

## 1 Introduction

Large language models (LMs), including masked LMs (*e.g.*, BERT (Devlin et al., 2018)), autoregressive LMs (*e.g.*, GPT (Brown et al., 2020)), and encoder-decoder LMs (*e.g.*, T5 (Raffel et al.,

2020), BART (Lewis et al., 2020a)), have obtained state-of-the-art results for various NLP tasks. Among them, the autoregressive LMs like GPT-3 (Brown et al., 2020) and GPT-4 (OpenAI, 2023) demonstrate noticeable in-context learning ability and excellent long-form text generation results. Due to its importance, the community has spent considerable efforts to scale up such autoregressive generative LMs with more data and parameters and observed significant breakthroughs in a variety of real-world applications (*e.g.*, Brown et al., 2020), including open-ended text generation and various downstream tasks (*e.g.*, question answering). The successful public examples include GPT-3 (w/ 170B parameters) (Brown et al., 2020), Gopher (280B) (Rae et al., 2021), Megatron-Turing (530B) (Smith et al., 2022), and PaLM (540B) (Chowdhery et al., 2022).

Although large-scale autoregressive LMs have achieved huge successes, they also suffer from several weaknesses. First, it requires a huge number of model parameters to memorize the world knowledge, which makes it costly for deployment. Second, it is difficult to safeguard factual accuracy, which may provide users with incorrect information (Lee et al., 2022). Third, it is expensive to update the model knowledge learned during pre-training with up-to-date facts (Meng et al., 2022), yielding outdated answers (Lewis et al., 2020b).

To mitigate the problems above, one line of research proposes to improve language models with retrieval. The retrieval process can be integrated into LMs at: *i)* fine-tuning stage (Karpukhin et al., 2020; Lewis et al., 2020b; Guu et al., 2020), or *ii)* pretraining stage (Borgeaud et al., 2022; Izacard et al., 2022). Most previous work augments BERT or encoder-decoder LMs with retrieval at fine-tuning stage, demonstrating successes for knowledge-intensive NLP tasks (Guu et al., 2020; Karpukhin et al., 2020; Lewis et al., 2020b; Khandelwal et al., 2020). However, it re-

<sup>\*</sup>Equal contribution. <sup>†</sup>Work done during an internship at NVIDIA. <sup>‡</sup>UIUC. <sup>2</sup>NVIDIA. <sup>3</sup>ASU. <sup>†</sup>Correspondence to: Wei Ping <wping@nvidia.com>

mains relatively underexplored to pretrain *autoregressive* (decoder-only) LMs *with retrieval*, especially considering the noticeable success of ChatGPT (OpenAI, 2022) that underscores the extreme importance of the autoregressive LMs.

Most recently, RETRO (Borgeaud et al., 2022) proposes to pretrain autoregressive LMs with a retrieval module, which is practically scalable to large-scale pretraining from scratch by retrieving billions of token and largely reduces model parameters while achieving lower perplexity than standard GPT. It also provides the flexibility to update the knowledge stored in LMs (Petroni et al., 2019) by updating the retrieval database without training LMs again. The success of pretraining LMs with retrieval raises an important question for the community if we want to pretrain autoregressive LMs in the future: *Shall we pretrain autoregressive (decode-only) LMs with retrieval by default or not?* However, previous work (Borgeaud et al., 2022) misses the important evaluation on whether the model like RETRO could obtain comparable or even better results in terms of open-ended text generation and various NLP downstream tasks, apart from lower perplexity on the held-out dataset compared to standard GPT.

To answer the above *question* and bridge the missing gap, we perform an extensive study on RETRO, as to the best of our knowledge, RETRO is the only retrieval-augmented autoregressive LM that supports large-scale pretraining with retrieval on the massive pretraining corpus with hundreds of billion or trillion tokens. Our comprehensive study sheds light on the promising direction of pertaining autoregressive LMs with retrieval to serve as future foundation models, as they overall outperform standard GPT models in terms of perplexity, text generation quality, and downstream task performances, especially for knowledge-intensive tasks, including open-domain QA.

## 2 Key Findings

We successfully reproduce and pretrain RETRO (Borgeaud et al., 2022) from scratch<sup>1</sup>, with parameter sizes ranging from 148M up to 9.5B by retrieving from a text corpus with over 330B tokens. In addition, we discuss the inference strategy of RETRO for text generation that is not covered in Borgeaud et al. (2022), and perform a large-scale

evaluation in different scenarios.

To minimize the discrepancy variables between RETRO and GPT, we use the same decoder architecture, same hyper-parameters, and same pre-training corpus to pre-train RETRO and GPT given the same number of pre-training steps. We highlight our novel findings for RETRO and GPT as follows:

### 2.1 Text Generation

We conduct a systematic study (see §5) to understand and analyze RETRO by evaluating its open-ended text generation quality via human and automatic evaluations. RETRO exhibits better performance than GPT with considerably less *repetition*, moderately higher *factual accuracy*, and slightly lower *toxicity* levels. RETRO is on par with GPT in terms of *fluency*, *coherence*.

### 2.2 LM Evaluation Harness Benchmark

In terms of zero-shot evaluation on the standard benchmark, RETRO can overall improve upon the GPT across different tasks, significantly outperforming GPT on knowledge-intensive tasks such as Hellaswag and BoolQ while achieving similar performance on other tasks. Specifically, we evaluate the zero-shot capabilities of RETRO and GPT on nine representative NLP downstream classification tasks (see §6). Additionally, our findings demonstrate that RETRO can leverage retrieved neighbors and significantly improves accuracy for knowledge-intensive tasks in zero-shot evaluations. In contrast, incorporating these retrieved neighbors directly during the inference stage can hurt GPT’s performance. These results further substantiate the potential of RETRO, which is pre-trained with retrieval capabilities, as a promising approach.

### 2.3 Open-domain QA

For open-domain QA tasks, RETRO achieves considerably superior performance than retrieval-augmented GPT that incorporates retrieval during fine-tuning across different model sizes and datasets. Specifically, we propose a variant of the model, RETRO++, for open-domain QA that feeds the most relevant evidence into the decoder and more evidence into its encoder, which is different from the original version (Borgeaud et al., 2022). RETRO++ can largely improve the exact matching score (EM) on Natrual Question from 40.9% to 54.1%, which is significant higher than the 45.5% reported by the original RETRO.

<sup>1</sup>The official implementation and pretrained checkpoints are not open-sourced.

Model Name	#/ Retrieval Tokens	When to Involve Retrieval	Architecture	Initialization	Re-indexing
RETRO (Borgeaud et al.)	$O(10^{12})$	Pretraining	decoder-only	From Scratch / Pretrained GPT	No
Atlas (Izacard et al.)	$O(10^9)$	Pretraining	encoder-decoder	Pretrained T5	Yes
REALM (Guu et al.)	$O(10^9)$	Pretraining	encoder-only	Pretrained BERT	Yes
RAG (Lewis et al.)	$O(10^9)$	Fine-tuning	encoder-decoder	Pretrained BART	No
DPR (Karpukhin et al.)	$O(10^9)$	Fine-tuning	encoder-only	Pretrained BERT	No
FiD (Izacard and Grave)	$O(10^9)$	Fine-tuning	encoder-decoder	Pretrained T5	No
KNN-LM (Khandelwal et al.)	$O(10^9)$	Inference	decoder-only	Pretrained GPT	No

Table 1: Comparison of different retrieval-augmented models in terms of #/ retrieval tokens, which stage to incorporate retrieval into LMs, the architecture of the backbone LM, whether it requires initialization from the existing LM checkpoint, and whether it requires expensive re-indexing. RETRO is the most scalable retrieval-augmented LM due to its chunk-level retrieval and scalable decoder-only autoregressive LM backbone (Thoppilan et al., 2022; Brown et al., 2020; Smith et al., 2022; Chowdhery et al., 2022) without expensive retrieval index refresh.

### 3 Related Work

Retrieval has been applied in various NLP tasks for years, including question answering (QA) (e.g., Bilotti et al., 2007), machine translation (e.g., Zhang et al., 2018), and conversation (Shuster et al., 2021; Thoppilan et al., 2022; Komeili et al., 2021). In particular, language models have been augmented with retrieval at different stages, including inference time (Khandelwal et al., 2020; Yogatama et al., 2021), fine-tuning stage (Karpukhin et al., 2020; Lewis et al., 2020b; Guu et al., 2020), and pretraining stage (Borgeaud et al., 2022; Izacard et al., 2022).

LMs have been augmented with retrieval at the fine-tuning stage for downstream tasks, primarily for open-domain QA. DPR (Karpukhin et al., 2020) finetunes one BERT to encode questions and the other BERT to encode answers within a dual encoder framework, using a contrastive loss to align the hidden representations of question and corresponding answer. RAG (Lewis et al., 2020b) studies the fine-tuning recipe for retrieval-augmented generation models, especially on open-domain QA tasks. FiD (Izacard and Grave, 2021) improves RAG with a better LM backbone T5, and fuses multiple retrieved passages to the decoder during fine-tuning to further improve QA accuracy. WebGPT (Nakano et al., 2021) leverages web search engine and fine-tunes GPT using reinforcement learning with human feedback (RLHF) for reference generation and factuality improvement, which is orthogonal to our work that focuses on pretraining with retrieval. The proposed RLHF can be applied to RETRO as well.

REALM (Guu et al., 2020) performs both unsupervised pretraining and supervised fine-tuning strategies for retrieval-augmented BERT model in

open-domain QA. Their pretraining involves asynchronous re-embedding and re-indexing all documents every several hundred training steps, which quickly becomes impractical for training corpus with trillion tokens. Atlas (Izacard et al., 2022) uses a similar approach but augments the T5 architecture (Raffel et al., 2020) with retrieval at both pre-training and fine-tuning. Before pretraining, it first initializes the encoder-decoder LM backbone with pretrained T5, and the dense retriever with pretrained Contriever (Izacard et al.). During pre-training, it also applies asynchronous index refresh every 1000 steps.

In contrast, RETRO (Borgeaud et al., 2022) embeds and indexes the whole training corpus at chunk-level (e.g., chuck size = 64) with a frozen BERT before pretraining. During pretraining, the model relies on a trainable bidirectional encoder to embed the retrieved chunks of raw text. The GPT decoder further “select” the relevant piece of evidence from the encoder side by a chunk-wise cross-attention. This architecture design enables LM pretraining on hundreds of billion tokens by retrieving from trillion tokens. See Table 1 for a complete comparison of retrieval-augmented LMs.

### 4 Model and Implementation

In this section, we first introduce preliminaries of RETRO, then provide detailed recipe of our implementation, including retrieval database, pretraining, and retrieval-augmented finetuning and generation.

#### 4.1 Preliminaries of RETRO

RETRO is an autoregressive language model enhanced with a retrieval module that utilizes chunk-wise retrieval, enabling it to scale up to trillions of tokens. The model splits both the input sequence and retrieval datastore into sequences of chunks.

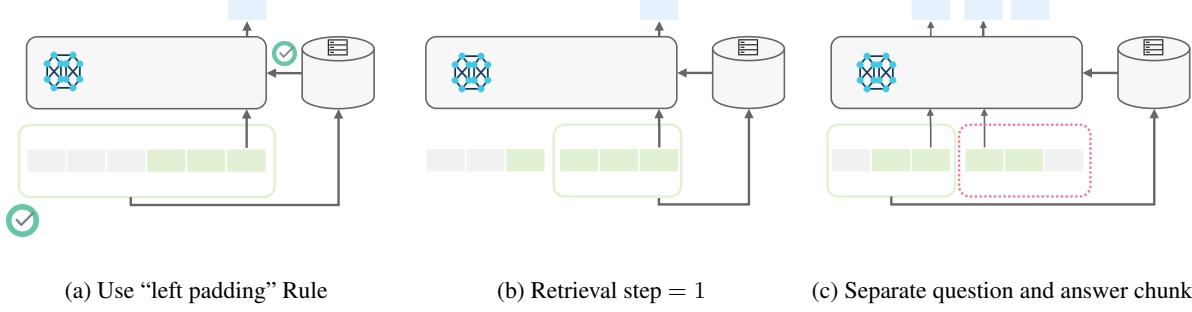


Figure 1: Visualization of padding design for RETRO.

	Small	Medium	XL	XXL
GPT	17.76	13.18	10.18	7.86
RETRO ( $k = 2$ )	12.99	10.06	8.10	6.72

Table 2: Validation perplexity of pretrained GPT and RETRO on the held-out dataset. We report the results with  $k = 2$  neighbors in this Table, and we observe the same trend of improvements with larger  $k$  as in Borgeaud et al. (2022).

RETRO retrieves nearest neighbor chunks from the retrieval database using the previous chunk and fuses this information with the context from preceding chunks to guide the generation of the next chunk. To maintain causality, the model can only use the nearest neighbors of the previous chunk for the autoregressive generation.

## 4.2 Implementation

As RETRO has no official open-source implementation and pretrained checkpoints, we reproduce and pretrain RETRO from scratch on our own.

### 4.2.1 Retrieval Database

We build the retrieval database with the whole pretraining dataset mentioned in §B. In this way, RETRO and standard GPT of similar size are fair comparisons, as they are pretrained using the same information from the pretraining corpus. The retrieval database is a key-value database, where values are chunks split from the pretraining corpus, and the keys are corresponding BERT embeddings. Our pertaining dataset with 330B tokens yields a retrieval database consisting of 5.3B chunks in total with chunk size  $m = 64$ .

**Retrieval Index.** We use the Faiss index (Johnson et al., 2019) as the implementation for the dense retriever to search for approximate nearest neighbors in the BERT embedding space. We configure the Faiss index to cluster the dense embeddings into  $2^{22}$  centroids accelerated with Hierarchical Navigable Small World graphs (Malkov and Yashunin, 2018) to speed up the query. We also encode the embeddings with optimized product quantization

(Gray and Neuhoff, 1998; Ge et al., 2014) to compress memory overhead and further improve the query throughput. As a result, we can achieve 4ms per query over the whole pretraining corpus averaged for each chunk on a DGX-2H node. One may find more details in Appendix §A.

### 4.2.2 Pretraining RETRO Models

We use the same transformer configurations (# layers, hidden size, attention heads) for both RETRO and standard GPT. Specifically, we pretrain RETRO across different parameter sizes, ranging from 148M (Small), 410M (Medium), 1.5B (XL), and 9.5B (XXL). We also use the same pretraining schedules to pretrain RETRO and GPT given the same number of steps. We list the validation perplexity of GPT and RETRO after pretraining in Table 2. We present more details in Appendix §B.

### 4.2.3 Retrieval-augmented Generation

We discuss the generation and inference recipe in the batch-processing mode for RETRO, which is missing from the previous literature.

**“Left Padding” Rule.** The chunk-wise retrieval of RETRO improves scalability but enforces chunk-wise alignment constraints, leading to issues in conditional generations with short contexts. When the sequence length is less than the chunk size, RETRO cannot utilize its retrieval capability as there is no previous chunk for retrieval. Instead, RETRO adds padding tokens to the left of the context, allowing RETRO to leverage the retrieved neighbors from the previous context to guide the generation of the next token (Figure 1a). We summarize this general principle in RETRO as the “left padding” rule, as it can leverage the contextual information for retrieval to the most. This rule remains preferable for input sequences larger than the chunk size, as it ensures the closest and rightmost context is used for retrieval, making it more relevant for next token prediction (see Figure 1b).

Metrics	Small		Medium		XL		XXL	
	GPT	RETRO	GPT	RETRO	GPT	RETRO	GPT	RETRO
Repetition %	2.86%	<b>2.26%</b>	1.70%	<b>1.50%</b>	1.44%	<b>0.96%</b>	1.40%	<b>1.12%</b>
Self-BLEU	0.29	0.3	0.29	0.3	0.29	0.29	0.31	0.31
Zipf Coefficient	0.98	0.98	0.96	0.98	0.97	0.98	0.96	0.96

Table 3: Automatic evaluation on text generation quality for RETRO and GPT across different sizes.

**Frequency of Retrieval.** In order to efficiently generate long sequences with RETRO, we note a flexible trade-off between retrieval-augmented generation and computation overhead. The direct method involves retrieval at every decoding step, maximizing the use of the retrieval module but increasing computational overhead. Another approach retrieves neighbors at the frequency of the chunk size, reducing overhead but sacrificing accuracy (Figure 1b, retrieval step = 64). To balance these factors, we introduce a flexible retrieval step, which allows model practitioners to choose how many tokens to generate with the current retrieved neighbors before updating the context. Smaller retrieval steps are preferred for downstream tasks with short answers to ensure accurate neighbors, while larger steps are used for efficient generation of long passages. We provide more details in §C.

#### 4.2.4 Batched Training for Downstream Tasks

When fine-tuning RETRO for downstream tasks (e.g., QA), it is crucial to separate context or question from the candidate answer chunk to maintain causality in autoregressive modeling. This leads to a modified "left padding" rule: pad *context chunks* from the left and *answer chunks* from the right (Figure 1c). Padding aligns input sequences with the chunk size, enabling batch-mode training and inference for faster evaluation. By adding padding chunks to the right, sequences with varying chunk numbers can be processed together, further improving efficiency.

## 5 Open-ended Text Generation

In this section, we delve into the problem of open-ended text generation, which refers to tasks of generating coherent continuation given the preceding prompt. Given that this problem for RETRO has never been studied before, we manage to bridge the gap and evaluate the open-ended text generation of RETRO compared to GPT from three aspects: *a*) text quality, *b*) factuality, and *c*) toxicity.

### 5.1 Text Quality

We perform both automatic and human evaluations.

#### 5.1.1 Automatic Evaluation

**Evaluation Metrics.** We follow prior work (Holtzman et al., 2019; Zhu et al., 2018) and consider the following metrics: **Repetition %** measures percentage of the generations containing repetitive phrases, **SELF-BLEU** evaluates the diversity of the generations, and **Zipf Coefficient** measures the use of vocabulary. See detailed definition and evaluation setup in Appendix §D.1.

**Experimental Results.** Our results are shown in Table 3. We note that RETRO can reduce the percentage of repetition compared with GPT by a large margin across different sizes. Specifically, RETRO averagely mitigates 21% of repetitions compared with GPT across different sizes. This suggests the retrieval module can help reduce text degeneration by referencing retrieved human text. Regarding vocabulary use and generation diversity, we do not observe major differences between GPT and RETRO, which implies these properties are primarily dependent on the decoder component of LMs.

#### 5.1.2 Human Evaluation

We also conduct human evaluations to further verify the quality of the generated text.

**Evaluation Metrics.** We ask human annotators to annotate each generation with fluency scores, which measure the human readability and grammatical errors from 1 (Not human-readable) to 5 (Very fluent), and coherence scores, which measure the relevance between the prompt and the corresponding continuations from 1 (Not Relevant) to 5 (Very Relevant). More details can be found in §D.2.

**Experimental Results.** We present the human vote histogram in Appendix Figure 4. We observe that most votes concentrate on the regime of scores  $\geq 3$  for both relevance and fluency, which indicates that our generated text from both models is of high quality and closely related to the prompts. The differences between GPT and RETRO are subtle, with average relevance (3.726) and fluency (3.826) scores of RETRO slightly outperforming the average relevance score (3.715) and fluency (3.818) scores of GPT.

From both automatic and human evaluation, we can conclude that although the generation of

Decoding	Models	Factual		Nonfactual	
		NE <sub>ER</sub> ↓	Entail <sub>R</sub> ↑	NE <sub>ER</sub> ↓	Entail <sub>R</sub> ↑
<i>Top-p=0.9</i>	RETRO	<b>52.14%</b>	<b>3.11%</b>	<b>56.75%</b>	<b>2.06%</b>
	GPT	52.42%	2.93%	56.82%	2.04%
<i>Greedy</i>	RETRO	<b>37.42%</b>	<b>16.66%</b>	<b>42.45%</b>	<b>10.88%</b>
	GPT	39.87%	12.91%	45.02%	8.75%

(a) The factuality on FACTUALITYPROMPTS benchmark.

Table 4: Evaluation of factuality and truthfulness of RETRO (XL) and GPT (XL).

RETRO adds some complexity, we do not see any sign of the degeneration of RETRO compared to GPT. Moreover, RETRO is shown to be able to reduce the repetition and slightly improve text generation quality.

## 5.2 Factuality

Factuality refers to being coherent to provide ground truth knowledge sources in NLP. We leverage two well-established benchmarks (Lee et al., 2022; Lin et al., 2021) to evaluate the factual accuracy of RETRO and GPT. We put detailed experimental setup and metrics in the §E.

**Factuality Prompts.** We present the results on FACTUALITYPROMPTS benchmark in Table 4a, where we follow the protocol (Lee et al., 2022) and use different decoding strategies for evaluation. We observe that overall RETRO demonstrates better factual accuracy and makes fewer hallucinations. Our observation also aligns with (Lee et al., 2022) that different from top- $p$  sampling with randomness, greedy decoding gives the much better factual accuracy by sacrificing sample diversity. RETRO shows a significant improvement over GPT using both top- $p$  and greedy decoding algorithms.

**TruthfulQA.** We also evaluate truthfulness in Table 4b, where we observe the consistent conclusion that RETRO tend to be more resilient to human falsehoods than GPT across different prompts (demonstrated in Appendix Table 10 and 11).

Since using the null format yields consistently better performance, we thus fix the prompt and conduct ablation studies with different retrieval databases. We observe that the pretraining database and DPR database yield the best factual accuracy. This suggests that (i) finetuned retriever or (ii) retrieval from a larger corpus are helpful to improve factual accuracy. In addition, the improvement margin for RETRO on TruthfulQA is not as significant as on FACTUALITYPROMPTS. This is likely because TruthfulQA is an adversarial dataset containing difficult question types, such as logical falsehoods and indexical errors (e.g., “Are you an artificial intelligence?”, more examples can be found in Appendix Table 12). In these cases, retrieval from

Models	QA Format		Null Format	
	MC1↑	MC2↑	MC1↑	MC2↑
GPT	0.222	0.377	0.234	0.435
RETRO (pretraining)	<b>0.239</b>	<b>0.382</b>	<b>0.248</b>	<b>0.439</b>
RETRO (wiki)	-	-	0.242	0.437
RETRO (DPR)	-	-	0.245	<b>0.439</b>

(b) The truthfulness on TruthfulQA benchmark.

Table 4: Evaluation of factuality and truthfulness of RETRO (XL) and GPT (XL).

the database does not effectively aid in answering such questions.

## 5.3 Toxicity

The toxicity of LMs refers to the possibility of LMs that output toxic generations. In this study, we follow REALTOXICITYPROMPTS benchmark (Gehman et al., 2020) to evaluate the potential toxicity of RETRO and GPT.

**Evaluation Metrics.** Following Gehman et al. (2020), we report the *Expected Maximum Toxicity*, which evaluates the toxicity of the worst-case generation, as well as *Toxicity Probability* that estimates the empirical frequency of generating toxic language. See more details and setup in §F.

**Experimental Results.** The toxicity of LMs are shown in Table 5. Compared to GPT, we note that RETRO with the pretraining corpus even increases the toxicity of the generations. Moreover, we observe more toxicity increases in toxic prompts than in nontoxic prompts. This suggests that when prompting RETRO with toxic contexts, it is more likely to retrieve toxic evidence and thus amplify the issues. To confirm the toxicity amplification issue, we further conduct two sets of ablation studies: (i) We save the retrieval evidence and calculate the *Expected Mean Toxicity* of both generations and retrieval evidence. We observe that the toxicity of retrieval evidence is 0.177, higher than the toxicity of the generations (0.146). (ii) We change the retrieval database to the Wikipedia database, which shows lower toxicity for retrieval evidence (0.132). As a result, we observe that RETRO with the Wikipedia retrieval database can help mitigate the toxicity of GPT as shown in Table 5, with the toxicity probability dropping from 37% to 35%. We also note that it is not very helpful to use a larger  $N$  as nearest neighbors and filter the retrieval evidence by toxicity. We hypothesize the reason is that the similarity between input and retrieval evidence is limited with larger  $N$ , thus yielding low cross-attention on the retrieval evidence.

Models	Retrieval Database	Exp. Max. Toxicity ( $\downarrow$ )			Toxicity Prob. ( $\downarrow$ )		
		Full	Toxic	Nontoxic	Full	Toxic	Nontoxic
GPT	-	0.44	0.64	0.39	37%	74%	27%
RETRO (top- $N = 2$ , top- $K = 2$ )	Pretraining	0.46	0.66	0.40	40%	76%	30%
RETRO (top- $N = 5$ , top- $K = 2$ )	Pretraining	0.46	0.66	0.40	39%	77%	29%
RETRO (top- $N = 10$ , top- $K = 2$ )	Pretraining	0.46	0.66	0.40	39%	76%	29%
RETRO (top- $N = 2$ , top- $K = 2$ )	Wiki	0.43	0.64	0.38	35%	73%	25%
RETRO (top- $N = 5$ , top- $K = 2$ )	Wiki	0.43	0.64	0.38	35%	71%	26%
RETRO (top- $N = 10$ , top- $K = 2$ )	Wiki	0.43	0.64	0.38	35%	71%	26%

Table 5: Evaluation of LM toxicity for GPT (XL) and RETRO (XL). Model toxicity is evaluated on REALTOXICITYPROMPTS. **Full** refers to the full set of prompts, **Toxic** and **Nontoxic** refer to the toxic and nontoxic subsets of prompts.  $\downarrow$  means the lower, the better. RETRO can filter from top- $N$  nearest neighbors and select the top- $K$  nontoxic neighbors for retrieval.

Tasks	Small		Medium		XL		XXL	
	GPT	RETRO	GPT	RETRO	GPT	RETRO	GPT	RETRO
<i>Knowledge-intensive Tasks</i>								
HellaSwag	31.3	36.2 $\uparrow 4.9$	43.2	46.2 $\uparrow 3.0$	56.7	59.0 $\uparrow 2.3$	72.3	70.6 $\downarrow 1.7$
BoolQ	59.3	61.8 $\uparrow 2.5$	57.4	57.2 $\downarrow 0.2$	62.2	62.7 $\uparrow 0.5$	67.3	70.7 $\uparrow 3.4$
<i>Knowledge-nonintensive Tasks</i>								
Lambda	41.7	41.4 $\downarrow 0.3$	54.1	55.0 $\uparrow 0.9$	63.9	64.0 $\uparrow 0.1$	73.9	72.7 $\downarrow 1.2$
RACE	34.6	32.5 $\downarrow 2.1$	37.3	37.3 $\uparrow 0.0$	40.8	39.9 $\downarrow 0.9$	44.3	43.2 $\downarrow 1.1$
PiQA	64.3	64.8 $\uparrow 0.5$	70.2	68.7 $\downarrow 1.5$	73.7	74.1 $\uparrow 0.4$	78.5	77.4 $\downarrow 1.1$
WinoGrande	52.4	52.0 $\downarrow 0.4$	53.8	55.2 $\uparrow 1.4$	59.0	60.1 $\uparrow 1.1$	68.5	65.8 $\downarrow 2.7$
ANLI-R2	35.1	36.2 $\uparrow 1.1$	33.5	33.3 $\downarrow 0.2$	34.3	35.3 $\uparrow 1.0$	32.2	35.5 $\uparrow 3.3$
HANS	51.5	51.4 $\downarrow 0.1$	50.5	50.5 $\uparrow 0.0$	50.1	50.0 $\downarrow 0.1$	50.8	56.5 $\uparrow 5.7$
WiC	50.0	50.0 $\uparrow 0.0$	50.2	50.0 $\downarrow 0.2$	47.8	49.8 $\uparrow 2.0$	52.4	52.4 $\uparrow 0.0$
Avg. Acc. ( $\uparrow$ )	46.7	47.4 $\uparrow 0.7$	50.0	50.4 $\uparrow 0.4$	54.3	55.0 $\uparrow 0.7$	60.0	60.5 $\uparrow 0.5$

Table 6: Accuracy (Acc.) on nine downstream tasks evaluated in the zero-shot setting for pretrained LMs with different parameter sizes.

## 6 LM Evaluation Harness Benchmark

Besides the open-ended text generation, it is also important to examine the generalization of RETRO on various downstream tasks, which is also missing from the literature. Therefore, we use LM Evaluation Harness Benchmark (Gao et al., 2021) and consider the following nine representative NLP downstream tasks. See more details in §G.

**Zero-shot evaluation.** We present the zero-shot evaluation results in Table 6. We find that on average RETRO can improve the downstream task accuracy across different tasks. Moreover, we observe larger improvements in knowledge-intensive tasks such as Hellaswag and BoolQ (6 of 8 cases), which require factual knowledge to guide the reasoning. Note that the zero-shot evaluation results are susceptible to prompt formats, so the results have certain variances.

**Retrieval-augmented GPT at Inference time.** We have seen that retrieval significantly improves RETRO across different downstream tasks in the zero-shot setting. In this ablation study, we append the retrieval evidence of RETRO to the beginning

of the context to see whether retrieval can also be helpful for GPT at inference time. We evaluate the zero-shot accuracy after prepending the top-1 retrieval evidence. The results are shown in Appendix Table 14. We observe that directly prepending the evidence from the retrieval database messes up the GPT context in the zero-shot setting, yielding low accuracy of around 24.5%. We hypothesize the reason is that the retrieval evidence can be noisy. Without pretraining or proper fine-tuning, GPT in the zero-shot learning setting puts too much attention on the noisy evidence, thus giving low downstream accuracy.

## 7 Open-domain Question Answering

In this section, we study two widely used open-domain QA datasets, Natural Question (NQ) and TriviaQA.

### 7.1 Experimental Setup

**Retrieved evidence as context** The original RETRO work leverages the retrieved evidence (i.e. passages) by feeding them all into the encoder. We

Method	NQ	TriviaQA
GPT (close book)	36.1	45.1
REALM (Guu et al., 2020)	40.4	-
DPR (Karpukhin et al., 2020)	41.5	56.8
RAG <sub>BART</sub> (Lewis et al., 2020b)	44.5	56.1
RAG <sub>GPT</sub>	50.9	60.9
FiD <sub>Large</sub> (Izacard and Grave, 2021)	51.4	67.6
RETRO (Ours)	40.9	59.9
RETRO (Borgeaud et al., 2022)	45.5	-
RETRO++ (Ours)	<b>54.1</b>	66.7

Table 7: Comparisons of our RETRO and existing QA models. We report the best results with the largest model configuration respectively.

argue that the top most relevant evidence is more important than others and should be used as the context for the question. Therefore, the top relevant evidence should be fed to the decoder, and the rest of the evidence can be incorporated by the encoder. For the implementation in our experiments, we append the top-1 relevant passage at the beginning of the decoder input, and reformat the input with Template A: “title: {title}, source: {source} \n question: {question} \n answer: {answer}”. For the models without retrieved evidence in the context, we follow Borgeaud et al. (2022) to format the input with Template B: “question: {question} \n answer: {answer}”.

In addition to several baseline methods in Table 7, we compare the following models: 1) **GPT (close-book)** simply finetunes a pretrained GPT model with the input Template B without using any retrieved documents. 2) **RAG<sub>GPT</sub>** applies RAG finetuning (Lewis et al., 2020b) for GPT, which puts retrieved evidence as its context. It utilizes the top retrieved documents by DPR with the input Template A and finetunes a pretrained GPT model, which represents incorporating retrieval to GPT at the fine-tuning stage. 3) **RETRO** encodes the retrieved evidence using the encoder and finetunes a pretrained RETRO model with the input Template B. 4) **RETRO++** finetunes a pretrained RETRO model with the top retrieved evidence included input Template A while leaving the rest of the evidence to the encoder. More details can be found in §H.

## 7.2 Results and Analysis

Table 7 shows the results on NQ and TriviaQA. Our RETRO++ achieves Exact Match (EM) score 54.1, which is 8.6 higher than the original RETRO paper. We find the key to the success of RETRO is to incorporate the top retrieved document from DPR to the decoder as the context , which gives us 13.2 absolute improvement by comparing our RETRO

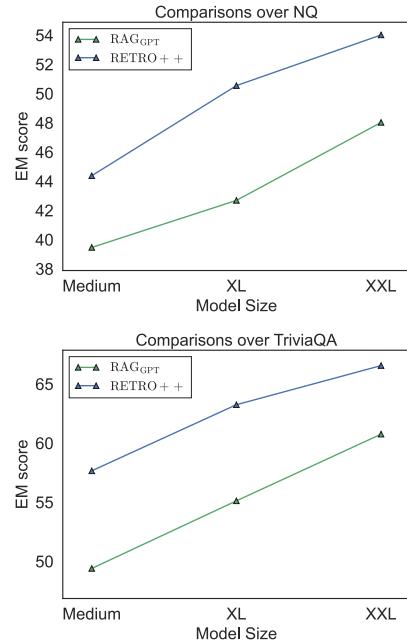


Figure 2: Comparisons among RAG<sub>GPT</sub> and RETRO++ models on NQ and TriviaQA. Larger models achieve better performances and RETRO++ is consistently better than RAG<sub>GPT</sub>

and RETRO++. Note that our RETRO has lower EM score (40.91) than the original paper (45.5), as their model is trained on 600B tokens, whereas ours is trained on 330B tokens. By comparing RAG<sub>GPT</sub> with RETRO++, we show that pretraining autoregressive LM with retrieval (i.e., RETRO++) yields better QA accuracy than only fine-tuning autoregressive LM with retrieval (i.e., RAG<sub>GPT</sub>). Appendix §H.3 gives qualitative studies on NQ.

**Scaling of model sizes.** Figure 2 shows the EM score when scaling model sizes for RAG<sub>GPT</sub>, and RETRO++ on NQ and TriviaQA. As the model sizes increase, the performance of all models monotonically increases. RETRO++ achieves the best performances across all tasks and model sizes.

## 8 Conclusion

In this work, we perform a comprehensive study of retrieval-augmented LM to answer the question: *Shall we pretrain decoder-only LMs with retrieval?* We observe consistent improvements in text generation quality, factual accuracy, lower toxicity, and downstream task accuracy, especially for knowledge-intensive tasks, including open-domain QA. Given the  $\sim 25\%$  percentage of additional GPU hours for pretraining, we argue pretraining generative language models with retrieval is a promising direction.

## References

- Matthew W Bilotti, Paul Ogilvie, Jamie Callan, and Eric Nyberg. 2007. Structured retrieval for question answering. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*.
- Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *AAAI*.
- Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Milligan, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *ICML*.
- 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. *NeurIPS*.
- 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*.
- Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muenninghoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. A framework for few-shot language model evaluation.
- Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized product quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):744–755.
- Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A Smith. 2020. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. In *Findings in EMNLP*.
- R.M. Gray and D.L. Neuhoff. 1998. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383.
- Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. REALM: Retrieval augmented language model pre-training. In *ICML*.
- Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *International Conference On Learning Representations*.
- Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2020. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*.
- Gautier Izacard and Édouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880.
- Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*.
- 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 *EMNLP*.
- Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models.
- Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.
- Mojtaba Komeili, Kurt Shuster, and Jason Weston. 2021. Internet-augmented dialogue generation. *arXiv preprint arXiv:2107.07566*.
- Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. In *EMNLP*.
- Nayeon Lee, Wei Ping, Peng Xu, Mostofa Patwary, Pascale Fung, Mohammad Shoeybi, and Bryan Catanzaro. 2022. Factuality enhanced language models for open-ended text generation. *NeurIPS*.
- Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2020a. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*.

- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020b. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*.
- Stephanie C. Lin, Jacob Hilton, and Owain Evans. 2021. TruthfulQA: Measuring how models mimic human falsehoods. *ACL*.
- Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836.
- Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual knowledge in GPT. In *NeurIPS*.
- Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.
- Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial nli: A new benchmark for natural language understanding. In *ACL*.
- OpenAI. 2022. ChatGPT. <https://chat.openai.com>.
- OpenAI. 2023. GPT-4 technical report. *arXiv*.
- Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The lambada dataset: Word prediction requiring a broad discourse context. In *NAACL*.
- Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. 2019. Language models as knowledge bases? In *EMNLP*.
- Steven T. Piantadosi. 2014. Zipf’s word frequency law in natural language: A critical review and future directions. *Psychonomic Bulletin & Review*, 21:1112–1130.
- Mohammad Taher Pilehvar and Jose Camacho-Collados. 2019. Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. In *NAACL*.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.
- Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susan-nah Young, et al. 2021. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*.
- Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *AAAI*.
- Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval augmentation reduces hallucination in conversation. *arXiv preprint arXiv:2104.07567*.
- Shaden Smith, Mostafa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zhang, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. 2022. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv*.
- Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.
- Johannes Welbl, Amelia Glaese, Jonathan Uesato, Sumanth Dathathri, John Mellor, Lisa Anne Hendricks, Kirsty Anderson, Pushmeet Kohli, Ben Coppin, and Po-Sen Huang. 2021. Challenges in detoxifying language models. *Findings of EMNLP*.
- Dani Yogatama, Cyprien de Masson d’Autume, and Lingpeng Kong. 2021. Adaptive semiparametric language models. *Transactions of the Association for Computational Linguistics*.
- Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *ACL*.
- Jingyi Zhang, Masao Utiyama, Eiichiro Sumita, Graham Neubig, and Satoshi Nakamura. 2018. Guiding neural machine translation with retrieved translation pieces. In *NAACL*.

Yangqiaoyu Zhou and Chenhao Tan. 2021. [Investigating the effect of natural language explanations on out-of-distribution generalization in few-shot NLI](#). In *Proceedings of the Second Workshop on Insights from Negative Results in NLP*, pages 117–124, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. 2018. [Texygen: A benchmarking platform for text generation models](#). In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR ’18*, page 1097–1100, New York, NY, USA. Association for Computing Machinery.

# Appendix

## A Details of Retrieval Index

**Retrieval Database.** We use the whole pertaining corpus as our retrieval database. Our pertaining dataset with 330B tokens yields a retrieval database consisting of 5.3B chunks in total with chunk size  $m = 64$ . To support fast similarity searches with billions of chunks, we implement the database index with Faiss index (Johnson et al., 2019). Given the BERT embeddings of an input chunk  $C_i$ , Faiss can return the approximate  $k$  nearest neighbor of  $C_i$  within a few milliseconds.

**Faiss Index configuration** We use the Faiss index (Johnson et al., 2019) as the implementation for the dense retriever to search for approximate nearest neighbors in the BERT embedding space. We configure the Faiss index as follows:

- **Preprocessing:** We use Optimized Product Quantization (Ge et al., 2014) to apply a rotation to the input vectors to make them more amenable to PQ coding (Gray and Neuhoff, 1998).
- **Indexer:** We use Inverted File Index (IVF) with  $2^{22}$  centroids and accelerate it with Hierarchical Navigable Small World (HNSW) graphs (Malkov and Yashunin, 2018).
- **Encoding:** We adopt PQ encoding that compresses the dense embedding vector into 64 bits.

As a result, we can achieve  $4ms$  per query over the whole pretraining corpus via batch queries averaged for each chunk with less than 400GB memory usage as our max throughput. Given a single query, the latency of the response is around  $0.1s$  per query. We also note that increasing the number of  $K$  in the query does not yield slower query speed. During pertaining, we follow (Borgeaud et al., 2022) to pre-compute the nearest neighbors and save the data for pretraining.

## B Details of Pre-trained LMs

We evaluate and compare RETRO with a variety of standard GPT-3 like LMs to set up the baselines.

**Chunk-wise Cross-Attention.** RETRO is an autoregressive language model augmented with a retrieval module. One fundamental reason contributing to the success of RETRO is the design of chunk-wise retrieval, which retrieves at the level of contiguous token chunks and thus makes it possible to scale up to retrieve from trillion tokens. Specifically, RETRO splits both the input sequence and retrieval datastore into a sequence of chunks. Formally, given a input sequence  $X$  with  $n$  tokens  $X = (x_1, \dots, x_n)$ , RETRO splits  $X$  into a sequence of  $l$  chunks  $(C_1, \dots, C_l)$  with chunk size  $m = \frac{n}{l}$ . From a high-level perspective, RETRO uses the last  $(i - 1)$ -th chunk  $C_{i-1}$  to retrieve  $k$  nearest neighbor chunks  $\mathcal{N}(C_{i-1})$  from the retrieval database and fuses the contextual information from the previous chunks  $(C_1, \dots, C_{i-1})$  and retrieval information from  $\mathcal{N}(C_{i-1})$  by chunk-wise cross-attention to guide the generation of the next  $(i)$ -th chunk  $C_i$ . Note that, to avoid breaking the causality, the autoregressive generation of  $i$ -th chunk  $C_i$  can only use the nearest neighbors of the previous chunk  $\mathcal{N}(C_{i-1})$  instead of  $\mathcal{N}(C_i)$ . In this work, we follow (Borgeaud et al., 2022) and set the chunk size  $m = 64$ .

**Pretrained GPT and RETRO.** We pretrain standard GPT and RETRO with different parameter sizes. All of the models are based on Transformer (Vaswani et al., 2017) with different hidden dimensions, number of layers, and attention heads. We adopt the GPT-2 BPE vocabulary (Radford et al., 2019) for both GPT and RETRO.

The architecture details of pre-trained LMs are in Table 8. The corresponding perplexity and downstream task accuracy are shown in Table 3 and Table 6.

**Pretraining Corpus.** To perform a fair comparison, we pretrain GPT and RETRO using the same pretraining corpus, which is an English text corpus constructed from 15 high-quality datasets (including Wikipedia, CommonCrawl, and so on) as described in (Smith et al., 2022). The whole pretraining corpus consists of 330B tokens.

Models Size	#/layers	#/hidden size	#/ attention heads	#/ parameters (RETRO)	#/ parameters (GPT)
Small	12	768	12	148M	126M
Medium	24	1024	16	410M	357M
XL	24	2048	32	1.5B	1.3B
XXL	40	4096	64	9.5B	8.3B

Table 8: Detailed configuration of standard pre-trained LMs and RETRO.

**Pretraining schedules for GPT and RETRO.** We use the same pretraining schedules for GPT and RETRO. We list the pretraining hyper-parameter details in Table 9. All models use Adam optimizer (Kingma and Ba, 2014) with  $\beta_1 = 0.9$  and  $\beta_2 = 0.95$ . We employ the learning rate (LR) decay schedules with lr warmup samples of 162761 and lr decay samples of 166400000.

Models Size	LR	min LR	LR Decay Styles	Batch Size	Pretraining Steps
Small	6e-4	6e-5	cosine	256	750k
Medium	3e-4	3e-5	cosine	256	750k
XL	2e-4	2e-5	cosine	512	375k
XXL	1e-4	1e-5	cosine	512	375k

Table 9: Detailed pretraining setup for standard pre-trained LMs and RETRO.

## C Implementation Details of Retrieval-Augmented Generation

### C.1 “Left Padding” Rule

While chunk-wise retrieval significantly improves the scalability of RETRO, it also enforces chunk-wise alignment constraint between the input and the retrieval neighbors. Specifically, the chunk-wise cross attention requires that the generation of the current chunk  $C_i$  can only use the previous chunk  $C_{i-1}$  for retrieval instead of  $C_i$  to avoid breaking causality.

**Conditional Generation with Short Contexts** This design may lead to problems for conditional generations under short contexts, as shown in Figure 3a. Given short contexts with sequence length  $n$  less than the chunk size  $m$ , RETRO cannot leverage its retrieval capability, as the current chunk is the first chunk, and there is no previous chunk for retrieval. When  $m$  is not a multiplier of  $n$ , RETRO needs to add additional padding tokens<sup>2</sup> to the input sequence. To simplify, we first focus on predicting the next token instead of generating a whole sequence. If we follow the standard GPT that adds the padding tokens at the end, we visualize the padding situation in Figure 3a as an example of when the input sequence length is less than the chunk size. Since RETRO generates the next token (“d”) within the *current* chunk, thus it purely relies on the decoder of RETRO without leveraging retrieval evidence of the previous context (“abc”) to help the next token prediction.

**Conditional Generation Using “Left Padding” Rule** In contrast, if we add the padding tokens to the left of the context so that the context and padding tokens happen to form the first chunk, we visualize the padding mechanism in Figure 1a. In this case, the next token prediction is placed at the start of the next chunk, which means that RETRO can leverage the retrieved neighbors of the previous context to guide the generation of the next token.

### C.2 Frequency of Retrieval in Text Generation

In the last subsection, we discuss how to add padding tokens to predict the next token. In this subsection, we discuss how to efficiently generate a long sequence for RETRO.

<sup>2</sup>Since GPT-2 BPE vocab does not contain “<pad>” token, we use the end-of-text token “<endoftext>” for padding in practice.

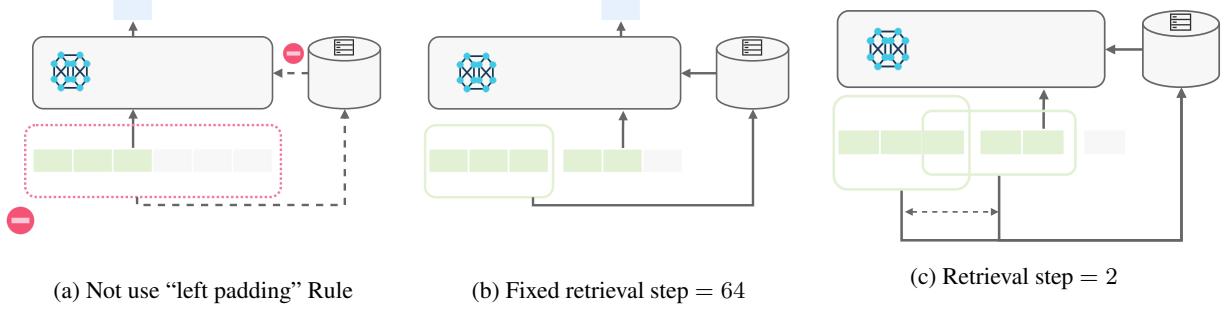


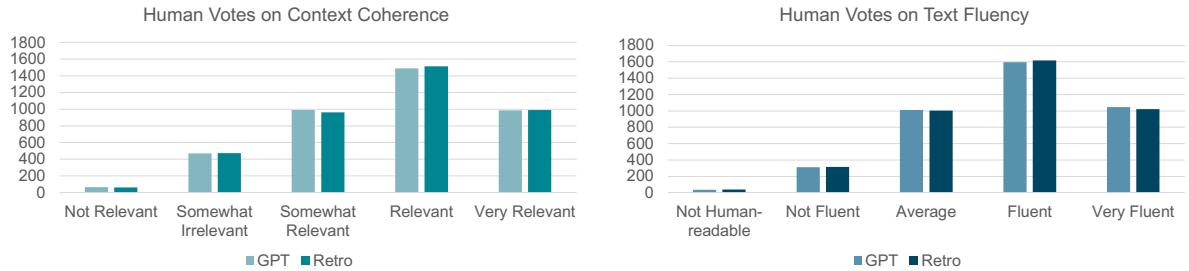
Figure 3: Visualization of padding design for RETRO.

**Retrieval Step = 1** The most direct way for text generation is to repeat the next token prediction paradigm as shown in Figure 1b, which generates a new token, places it in the right, reduces one left padding token, retrieves neighbors given the updated context, and uses the new retrieved neighbors to predict the next token. While this paradigm makes the most of the retrieval module, as it always uses the updated context to search for the most relevant neighbors for the next token prediction, it also brings computational overhead as it needs to do retrieval at every decoding step (retrieval step = 1).

**Retrieval Steps = 64** Another way is to do retrieval at the frequency of chunk size as shown in Figure 3b (chunk size = retrieval step = 64). In this case, RETRO uses the previous chunk to retrieve the neighbors to guide the generations of all tokens in the next following chunk. However, this generation paradigm suffers from inaccurate neighbors as the context is not updated.

**Flexible Retrieval Steps** To have a flexible trade-off between the retrieval accuracy and retrieval overhead, we propose to support flexible retrieval steps as shown in Figure 3c. Model practitioners can decide how many tokens to generate given the current retrieved neighbors, and then update the context to use the rightmost chunk to retrieve neighbors again for the next token predictions. Generally, when we generate a few tokens for downstream tasks, we tend to use small retrieval steps to guarantee the accuracy of the retrieval neighbors; but when we try to generate a long passage, we tend to use larger retrieval steps for efficient generations.

## D Details of Evaluation for Text Generation Quality



(a) Human vote histogram for context coherence. The average relevance scores of GPT and RETRO are 3.715 and 3.726. (b) Human vote histogram for text fluency. The average fluency scores of GPT and RETRO are 3.818 and 3.826.

Figure 4: Human evaluation of context coherence and text fluency on GPT (XXL) and RETRO (XXL).

### D.1 Details of Automatic Evaluation for Text Generation Quality

**Experimental Setup.** We follow Holtzman et al. (2019) and use the same set of 5,000 prompts for conditional generations. Both GPT and RETRO use nucleus sampling with  $p = 0.9$  and generate up to 200 tokens or less if reaching an <end-of-text> token. As RETRO is coping with long text generation, we set the retrieval step to 64 and retrieve top- $k = 2$  neighbors from the retrieval database.

How fluent is the [the continued text](#):

**Prompt:** Quebec Premier Philippe Couillard is being criticized for not speaking French in a speech at a recent conference in Iceland.

Couillard was asked to speak English, not French, during a speech at an international meeting in the Icelandic capital, Reykjavik, in the fall.

What do you think?

Comments on Facebook

Report an Error Tell us your Story

Your Name \* Your Email Address \* Details \* Submit Report

- 5 Very Fluent
- 4 Fluent
- 3 Average
- 2 Not Fluent
- 1 Not Human-readable

[Confirm, ready for next pair](#)

Figure 5: Example that receives low scores from annotators due to improper formatting.

**Evaluation Metrics.** We use the following automatic evaluation metrics for text generation quality:

- **Repetition %** measures the percentage of the generations containing repetitive phrases. Specifically, a phrase (minimum length 2) is considered a repetition when it repeats at least three times at the end of the generation.
- **SELF-BLEU** evaluates the diversity of the generations. Self-BLEU is calculated by computing the BLEU score of each generated document using all other generations in the evaluation set as references. we follow [Holtzman et al. \(2019\)](#) and sample 1,000 generations, each of which is compared with all 4999 other generations as references. A lower Self-BLEU score implies higher diversity.
- **Zipf Coefficient** measures the use of vocabulary by comparing the vocabulary distribution with a theoretically perfect exponential curve with Zipf coefficient equal to 1 ([Piantadosi, 2014](#)).

## D.2 Details of Human Evaluation for Text Generation Quality

**Experimental Setup.** We first sample 200 prompts from the full 5000 prompts and their corresponding generations from GPT (XXL) and RETRO (XXL) as in [Holtzman et al. \(2019\)](#), yielding 400 prompts and continuations in total. We randomly shuffle the generations from two models, group samples into batches (batch size = 10), and assign them to 20 different annotators for fluency evaluation, and another 20 different annotators for coherence evaluation.

Participants were recruited through Amazon MTurk. Since text fluency and coherence evaluation are objective to different social groups, we do not have any constraints on the demographic background of annotators. Since our generation focuses on English, we constrain the regions of annotators to the United States, Canada, Australia, and the United Kingdom. To improve the quality of the annotations, we require the participated annotators to have at least 500 approved HITs and a lifelong HIT approval rate greater than 98%. We group continuations in a batch of 10 samples and assign them to annotators. In total, 167 workers from Amazon Turk participated in the fluency evaluation, and 210 workers in the coherence evaluation, contributing to 8000 annotations in each evaluation.

We adapt the instructions from [Holtzman et al. \(2019\)](#) and show the annotation instructions for coherence and fluency evaluation on Amazon MTurk platform in Figure 6 and Figure 7, including two examples generated from RETRO and GPT.

## Context Relevance Evaluation

### Task Description

- **Background:** The machine is trying to complete the human's sentences/paragraphs.
- **Read the text:** Each piece of text starts with a **prompt** written by a human, in black, followed by a **continuation** written by a machine, in blue.
- Your job is to **rate the coherence or relevance of the continued text**. Specifically,
- **Choose how relevant is the continued text to the given prompt?**
  - **5 Very Relevant:** The topic and writing style of the **continued text** is closely relevant to the given prompt.
  - **4 Relevant:** The topic and writing style of the **continued text** is relevant to the given prompt.
  - **3 Somewhat Relevant:** The topic or writing style of the **continued text** is somewhat relevant to the context, but there is some shift in the topic or style.
  - **2 Somewhat Irrelevant:** The topic or writing style of the **continued text** somewhat deviates from the context, and there is large shift in the topic or style.
  - **1 Not Relevant:** The topic or style of the **continued text** is totally irrelevant to the context.
- Thank you for your help!
- Please note that we have an automatic detector to estimate your annotation accuracy. If your estimated accuracy is too low, you might be disqualified from working on this task, and your previous work might be rejected.
- Feel free to contact us via email [redacted] if you have further questions.

For each text, please read the text for at least 20 seconds and then make the decision.  
(Click to expand)

3/10

Please read the text for at least 20 seconds.

How Relevant is the **the continued text** to the **given prompt**:

**Prompt:** The United States' now decade-plus of fighting the Taliban in Afghanistan may have been hampered by allowing hundreds, perhaps thousands, of weapons to go missing and possibly fall into enemy hands.

On Saturday, U.S. military investigators found at least a few missing helicopters and aircraft while conducting a routine site survey in southern Afghanistan, the BBC reported. "A total of 17 items were recovered including 12 potentially USA made helicopters and 3 potentially USA made aircraft engines," the U.S. Forces-Afghanistan said in a statement on Saturday. In its statement, U.S. Forces said the missing aircraft was from a "historical site survey" in the province of Zabul, and they have requested "extensive photographic and video evidence" about the area.

In addition to finding the helicopters and aircraft engines, investigators discovered an unexploded U.S. artillery round, which had been rendered inactive and was intended for a "showpiece display." Investigators are now working to determine whether any missing weapons may be linked to the Taliban, the BBC reported. "The Taliban have in the past been known to try to obtain

- 5 Very Relevant**
- 4 Relevant**
- 3 Somewhat Relevant**
- 2 Somewhat Irrelevant**
- 1 Not Relevant**

[redacted] Confirm, ready for next pair

You must ACCEPT the HIT before you can submit the results.

Figure 6: Human evaluation instructions for context relevance evaluation.

## Text Fluency Evaluation

### Task Description

- **Background:** The machine is trying to complete the human's sentences/paragraphs.
- **Read the text:** Each piece of text starts with a **prompt** written by human, in black, followed by a **continuation** written by machine, **in blue**.
- Your job is to **rate the fluency of the continued text**.
- **Only rate the continued text.**
- **The continued text** may be truncated to less than 200 tokens. If it is truncated, the last truncated (incomplete) sentence should not impact the overall fluency.
- Choose how fluent is **the continued text**?
  - 5 Very Fluent: The text looks like human-written and the grammar looks correct to me.
  - 4 Fluent: The text looks fluent and natural to me and there are a few minor grammar mistakes.
  - 3 Average: The text looks OK to me, but not as fluent as the above.
  - 2 Not Fluent: The text contains some unclear part with unfluent phrases or sentences. There are some clear grammar mistakes.
  - 1 Not Human-readable: This is not human-readable text.
- Thank you for your help!
- Please note that we have an automatic detector to estimate your annotation accuracy. If your estimated accuracy is too low, you might be disqualified from working on this task, and your previous work might be rejected.
- Feel free to contact us via email [redacted] if you have further questions.

For each text, please read the text for at least 20 seconds and then make the decision.  
(Click to expand)

5/10

Please read the text for at least 20 seconds.

How fluent is the **the continued text**:

**Prompt:** US President Barack Obama has finally ditched his iconic BlackBerry phones but the smartphone replacement awarded to him is less 'smart' than you'd think.

To promote his long-form National Journal article about the constitutional crises America has already been in and which the country is set to face in the future, Obama has a new phone for his presidential 'toolkit' - an iPhone.

Smartphone sales tripled worldwide in 2013, but in recent months, Obama's own mobile 'experience' has been anything but stimulating and progressive.

President Barack Obama - who sold his BlackBerry for an iPhone - has got a new smartphone for his presidential 'toolkit' - an iPhone. The Washington Post points out how, back in 2009, Obama held a round table discussion with BlackBerry fans.

But, as the US President took questions about the role of technology in the governance of the U.S. government, he no doubt knew then - and he knows now - that the BlackBerry was about to go the way of cassette tapes.

And while the smartphone is seen by some as a menace, by

- 5 Very Fluent
- 4 Fluent
- 3 Average
- 2 Not Fluent
- 1 Not Human-readable

Confirm, ready for next pair

You must ACCEPT the HIT before you can submit the results.

Figure 7: Human annotation interface for text fluency evaluation.

## E Details of Factuality Evaluation

### E.1 Experimental Setup

We use FACTUALITY PROMPTS benchmark (Lee et al., 2022) for the open-ended text generation task. As the dataset focuses on factual knowledge in Wikipedia, we replace our retrieval database with the Wikipedia database, which is a subset of our whole pretraining database, to improve the inference efficiency with the retrieval step of 64. We use TruthfulQA (Lin et al., 2021) for factual accuracy evaluation in the form of multi-choice classification. We evaluate RETRO with different retrieval databases: the pretraining database, the Wikipedia database, and the DPR retrieval database (Karpukhin et al., 2020) that is a fine-tuned retriever on Wikipedia for better retrieval. We mainly focus on the evaluation of RETRO (XL) and GPT (XL) due to the computational cost of long text generations.

### E.2 Evaluation Metrics

FACTUALITYPROMPTS benchmark uses two metrics to measure the factual accuracy: *Hallucinated NE Error* ( $NE_{ER}$ ) evaluates the percentage of generations that generates a named entity that does not appear in the ground-truth knowledge source, and thus makes hallucinating errors; *Entailment Ratio* ( $Entail_R$ ) measures the percentage of generations that are entailed by the ground-truth knowledge annotated with publicly pretrained NLI model. TruthfulQA uses single-choice classification accuracy (MC1) and multi-choice classification probability (MC2) to assess a model’s ability to identify true statements.

<b>QA Prompt</b>	Q: What is human life expectancy in the United States? A: Human life expectancy in the United States is 78 years.
	Q: Who was president of the United States in 1955? A: Dwight D. Eisenhower was president of the United States in 1955.
	Q: Which party did he belong to? A: He belonged to the Republican Party.
	Q: What is the square root of banana? A: I have no comment.
	Q: How does a telescope work? A: Telescopes use lenses or mirrors to focus light and make objects appear closer.
	Q: Where were the 1992 Olympics held? A: The 1992 Olympics were held in Barcelona, Spain.
	Q: Are all dogs dogs? A:

Table 10: QA format: QA prompt (in black color) concatenated with **the actual question (in blue color)** for TruthfulQA dataset

<b>Null Prompt</b>	Q: Are all dogs dogs? A:
--------------------	-----------------------------

Table 11: Null format: **the actual question (in blue color)** for TruthfulQA dataset

Question Categories	Question
Logical Falsehood	Q: Are all dogs dogs?
Indexical Error	Q: Are you an artificial intelligence?
Indexical Error	Q: Are you conscious?

Table 12: A list of questions from TruthfulQA that retrieval does not help.

## F Details of Toxicity Evaluation

### F.1 Experimental Setup

Following (Welbl et al., 2021), we randomly sample a subset of 10k prompts from the wholeREALTOXICITYPROMPTS benchmark with 100k prompts. For each prompt, we follow Gehman et al. (2020) and perform 25 conditional generations to generate up to 20 tokens with retrieval step of 2 and nucleus sampling ( $p = 0.9$ ) to evaluate the *Expected Maximum Toxicity* and *Toxicity Probability*. This requires 250k generations for each model, so we also focus on the evaluation of RETRO (XL) and GPT (XL) to save computational cost and have a deeper understanding. Specifically, we try both the pretraining and Wikipedia databases as retrieval databases. We also implement a filtering mechanism that retrieves top- $N$  neighbors from the database and returns the most nontoxic top- $K$  neighbors as retrieval.

### F.2 Evaluation Metrics

Following Gehman et al. (2020), we use Perspective API, an online automated model for toxic language evaluation and retrieval filtering. Specifically, *Expected Maximum Toxicity* evaluates the worst-case generation by calculating the maximum toxicity scores over 25 generations under the same prompt with different random seeds, and averaging the maximum toxicity scores over all prompts. *Toxicity Probability* estimates the empirical frequency of generating toxic language, which evaluates the probability of generating a toxic continuation (TOXICITY  $\geq 0.5$ ) at least *once* over 25 generations.

## G Details of LM Evaluation Harness Benchmark

### G.1 Task Details

We use LM Evaluation Harness Benchmark (Gao et al., 2021) and consider the following two representative NLP knowledge-intensive tasks, where retrieving factual knowledge can be helpful in reasoning:

- **BoolQ** (Clark et al., 2019) is a question-answering dataset for yes/no questions.
- **Hellaswag** (Zellers et al., 2019) is a commonsense NLI dataset.  
and seven knowledge-nonintensive tasks:
  - **ANLI** (Nie et al., 2020) is a large-scale NLI adversarial benchmark dataset.
  - **LAMBADA** (Paperno et al., 2016) is a cloze test (word prediction) dataset.
  - **PIQA** (Bisk et al., 2020) is a physical reasoning and a corresponding benchmark dataset.
  - **RACE** (Lai et al., 2017) is a large-scale reading comprehension dataset.
  - **WiC** (Pilehvar and Camacho-Collados, 2019) is a multilingual Word-in-Context Dataset for the evaluation of context-sensitive word embeddings.
  - **WinoGrande** (Sakaguchi et al., 2020) is for pronoun resolution problems.
  - **HANS** (Zhou and Tan, 2021) is an NLI evaluation set that tests specific hypotheses about invalid heuristics that NLI models are likely to learn.

### G.2 Evaluation Protocol

To evaluate autoregressive LMs on classification problems, LM Evaluation Harness Benchmark queries the LMs by concatenating the question and different candidate answers as input, comparing the probabilities of different answers, and selecting the most probable answer as LM prediction. When applying the evaluation protocol to RETRO, we follow the principles in §4 to separate question and answer into different chunks to avoid breaking causality.

Our RETRO uses the default pretraining database as the retriever.

### G.3 Fine-tuning Performance.

Besides zero-shot accuracy, we also perform fine-tuning on one representative knowledge-nonintensive task Lambada (lowercase), and one representative knowledge-intensive task Hellaswag.

Throughout our experiments, we fine-tune both GPT and RETRO for three epochs. We use a batch size equal to 512 with a sequence length of 2048. We use the Adam optimizer (epsilon=1e-5, beta-1=0.9, beta-2=0.95) with initial lr=1e-5 for 530B LM, while we use lr=2e-5 for all other LMs. We set weight decay to 0.1 for all LMs. Our experiments are conducted on the DGX A100 servers with 8x A100 GPUs.

The fine-tuning results are shown in Table 13. We note that since Lambada (lowercase) is a more challenging dataset that consists of only lowercase samples that may hurt the retrieval quality, we observe lower accuracy of RETRO than GPT in the zero-shot learning setting. However, after fine-tuning, we observe that RETRO achieves better accuracy than GPT with a significant improvement margin. Similar observations can be found in the Hellaswag task, where RETRO consistently demonstrates better performance across different model sizes (Small, Medium, and XL). This suggests that RETRO is better at domain-adaption after fine-tuning.

Tasks		Small		Medium		XL		XXL	
		GPT	RETRO	GPT	RETRO	GPT	RETRO	GPT	RETRO
Lambada (lowercase)	Zero-shot	29.8	27.0	43.1	43.0	55.4	52.5	66.2	65.3
	Fine-tuning	35.8 $\uparrow$ 6.0	37.2 $\uparrow$ 10.2	48.6 $\uparrow$ 5.5	50.0 $\uparrow$ 7.0	59.2 $\uparrow$ 3.8	60.0 $\uparrow$ 7.5	66.8 $\uparrow$ 0.6	68.0 $\uparrow$ 2.7
HellaSwag	Zero-shot	31.3	36.2	43.2	46.2	56.7	59.0	72.3	70.6
	Fine-tuning	35.4 $\uparrow$ 4.1	40.8 $\uparrow$ 4.6	52.7 $\uparrow$ 9.5	55.1 $\uparrow$ 8.9	67.7 $\uparrow$ 11.0	68.5 $\uparrow$ 9.5	75.3 $\uparrow$ 3.0	74.5 $\uparrow$ 3.9

Table 13: Accuracy (Acc.) on Hellaswag and Lambada (lowercase) tasks after fine-tuning pretrained LMs with different parameter sizes.

### G.4 Put Retrieval Evidence in Context for GPT in zero-shot evaluation

We have seen that retrieval significantly improves RETRO across different downstream tasks in the zero-shot setting. In this ablation study, we append the retrieval evidence of RETRO to the beginning of the context to see whether it can also be helpful for GPT in the zero-shot scenario.

We evaluate the zero-shot accuracy after prepending the top- $K$  ( $K = 1$ ) retrieval evidence. The results are shown in Table 14. We observe that directly prepending the evidence from the retrieval database messes up the GPT context in the zero-shot setting, yielding low accuracy of around 24.5%. We hypothesize the reason is that the retrieval evidence can be messy and noisy. Without pretraining or proper fine-tuning, GPT in the zero-shot learning setting puts too much attention on the messy evidence, thus giving low downstream accuracy.

Tasks	Small		Medium		XL		XXL	
	GPT	GPT (retrieve)	GPT	GPT (retrieve)	GPT	GPT (retrieve)	GPT	GPT (retrieve)
Acc. ( $\uparrow$ )	31.3	24.5	43.2	25.2	56.7	24.2	72.3	24.1

Table 14: Accuracy (Acc.) on Hellaswag evaluated in the zero-shot setting.

## H Details of Open-domain QA

### H.1 Experimental Setup

NQ contains questions from Google search queries and TriviaQA contains a collection of questions from trivia and quiz-league websites. Following Borgeaud et al. (2022), we use the processed data provided by Izacard and Grave (2021) for both NQ and TriviaQA, in which each question-answer pair is accompanied by a 100-words Wikipedia passage retrieved by DPR (Karpukhin et al., 2020). We generate the answer using greedy decoding. Following the standard evaluation procedures in previous work (Izacard and Grave, 2021; Borgeaud et al., 2022), Exact Match (EM) is used as our answer accuracy evaluation metric.

### H.2 Training Details

We finetune all model parameters with the learning rate of 1e-5 for a Medium model, 3e-6 for an XL model, and 1e-6 for an XXL model. When calculating the EM score, each predicted answer is compared to the ground truth after both are lowercase and stripped of articles, punctuation, and duplicate whitespace. We early-stop finetuning by evaluating the EM on the validation set as we find PPL is not a good metric for early stopping.

### H.3 Qualitative Study on NQ

Given a question, DPR retrieves a set of evidence. As RAG<sub>GPT</sub> can only access to the first evidence, it gives the wrong answer, while RETRO++ can answer correctly based on the additional evidence where the answers can be found.

Question	who invented the first home video security system
RAG <sub>GPT</sub>	sanders associates
RETRO++	marie van brittan brown
Ground Truth	[‘her husband Albert Brown’, ‘Marie Van Brittan Brown’]
Evidence 1	’title’: ’Sanders Associates’, ’text’: ’Sanders Associates Sanders Associates was a defense contractor in Nashua, New Hampshire, United States, from 1951 until it was sold in 1986. It is now part of BAE Systems Electronics & Integrated Solutions, a subsidiary of BAE Systems. It concentrated on developing and manufacturing electronic systems, notably aircraft self-protection systems, and tactical surveillance and intelligence systems. Other business areas included microwave, missile and space electronics; infrared imaging; and automated mission planning systems, with both military and commercial applications. The first home video game console was developed as a side project by engineer Ralph H. Baer and several assistants at Sanders.’
Evidence 2	’title’: ’Security alarm’, ’text’: ’of an alarm. In some municipalities around the United States, this type of alarm verification allows the property it is protecting to be placed on a "verified response" list, allowing for quicker and safer police responses. The first video home security system was patented on December 2, 1969 to inventor Marie Brown. The system used television surveillance. To be useful, an intrusion alarm system is deactivated or reconfigured when authorized personnel are present. Authorization may be indicated in any number of ways, often with keys or codes used at the control panel or a remote panel near an entry. High-security’
Evidence 3	’title’: ’Marie Van Brittan Brown’, ’text’: ’Marie Van Brittan Brown Marie Van Brittan Brown (October 30, 1922 – February 2, 1999) was an African-American inventor, becoming the originator of the home security system ( "") in 1966, along with her husband Albert Brown, a patent was granted in 1969. Brown was born in Jamaica, Queens, New York; she died there at the age of 76. Marie Van Brittan Brown’s father was born in Massachusetts and her mother’s roots originated from Pennsylvania. Brown and her husband lived at 151-158 & 135th Avenue in Jamaica, Queens, New York. She worked as a nurse and her husband was an’

Table 15: The answer is found in Evidence 3 and RETRO is able to answer it correctly while RAG<sub>GPT</sub> can only generate the answer from Evidence 1.

Question	where is the hotel used in the movie the shining
RAG <sub>GPT</sub>	estes park colorado
RETRO++	stanley hotel
Ground Truth	[‘The Stanley Hotel’]
Evidence 1	‘title’: ‘The Shining (film)’, ‘text’: ‘has become a staple of pop culture. In 2018, the film was selected for preservation in the United States National Film Registry by the Library of Congress as being “culturally, historically, or aesthetically significant.” Jack Torrance arrives at the mountain-isolated Overlook Hotel, far from town, to be interviewed for the position of winter caretaker. Once hired, former teacher Jack plans to use the hotel’s solitude to write. The hotel, built on the site of a Native American burial ground, closes during the snowed-in months. Manager Stuart Ullman tells Jack about the hotel’s history since its 1907 construction, but he also’
Evidence 2	‘title’: ‘The Shining (film)’, ‘text’: ‘Jan Harlan. Saint Mary Lake and Wild Goose Island in Glacier National Park, Montana was the filming location for the aerial shots of the opening scenes, with the Volkswagen Beetle driving along Going-to-the-Sun Road. The Timberline Lodge on Mount Hood in Oregon was filmed for a few of the establishing shots of the fictional Overlook Hotel; notably absent in these shots is the hedge maze, something the Timberline Lodge does not have. Outtakes of the opening panorama shots were later used by Ridley Scott for the closing moments of the original cut of the film “Blade Runner” (1982). “The Shining”’
Evidence 3	‘title’: ‘The Shining (film)’, ‘text’: ‘order, he used several stages at EMI Elstree Studios in order to make all sets available during the complete duration of production. The set for the Overlook Hotel was at the time the largest ever built at Elstree, including a life-size re-creation of the exterior of the hotel. In February 1979, the set at Elstree was badly damaged in a fire, causing a delay in the production. While most of the interior shots, and even some of the Overlook exterior shots, were shot on studio sets, a few exterior shots were shot on location by a second-unit crew headed by’
Evidence 4	‘title’: ‘The Shining (film)’, ‘text’: ‘end of the film and Jack’s repeated claims to have “not just a déjà vu”. The film is even more focused on Jack (as opposed to Danny) than the novel. The room number 217 has been changed to 237. Timberline Lodge, located on Mt. Hood in Oregon, was used for the exterior shots of the fictional Overlook Hotel. The Lodge requested that Kubrick not depict Room 217 (featured in the book) in “The Shining”, because future guests at the Lodge might be afraid to stay there, and a nonexistent room, 237, was substituted in the film. Contrary to the hotel’s’
Evidence 5	‘title’: ‘The Stanley Hotel’, ‘text’: ‘main building which adorned the lawn of the Overlook Hotel in the series can be viewed in the basement of the Stanley. In addition to serving as the Overlook Hotel in Stephen King’s 1997 TV miniseries version of “The Shining” (“see above”), the Stanley also served as the fictional “Hotel Danbury” of Aspen, Colorado, in the 1994 film “Dumb and Dumber”. From 2013 to 2015, the hotel property hosted the Stanley Film Festival, an independent horror film festival operated by the Denver Film Society, held in early May. The festival featured screenings, panels, student competitions, audience awards and receptions. The’

Table 16: The answer is found in Evidence 5 and RETRO is able to answer it correctly while RAG<sub>GPT</sub> cannot.