Long-range Language Modeling with Self-retrieval

Ohad Rubin **Jonathan Berant**
The Blavatnik School of Computer Science, Tel Aviv University
{ohad.rubin,joberant}@cs.tau.ac.il

Abstract

Retrieval-augmented language models (LMs) have received much attention recently. However, typically the retriever is not trained jointly as a native component of the LM, but added to an already-pretrained LM, which limits the ability of the LM and the retriever to adapt to one another. In this work, we propose the *Retrieval-Pretrained Transformer* (RPT), an architecture and training procedure for jointly training a retrieval-augmented LM from scratch for the task of modeling long texts. Given a recently generated text chunk in a long document, the LM computes query representations, which are then used to retrieve earlier chunks in the document, located potentially tens of thousands of tokens before. Information from retrieved chunks is fused into the LM representations to predict the next target chunk. We train the retriever component with a semantic objective, where the goal is to retrieve chunks that increase the probability of the next chunk, according to a reference LM. We evaluate RPT on four long-range language modeling tasks, spanning books, code, and mathematical writing, and demonstrate that RPT improves retrieval quality and subsequently perplexity across the board compared to strong baselines.

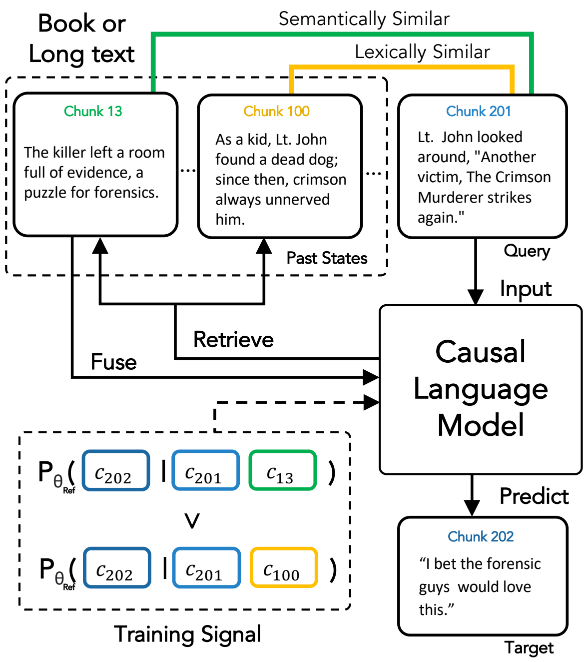

Figure 1: Retrieval-Pretrained Transformer (RPT) is a language model for long texts (e.g., books) trained from scratch with a native retrieval ability. RPT takes a chunk of text as input, retrieves semantically-relevant chunks from the past to better predict the next chunk, and fuses these retrieved chunks into its representations. On top of a standard LM loss, the retriever is trained to retrieve chunks that increase the probability of the next chunk according to a *reference LM*.

1 Introduction

Large language models (LMs) have had immense success recently (Brown et al., 2020; Chowdhery et al., 2022; Zhang et al., 2022; Touvron et al., 2023), becoming a useful tool across disciplines. However, their success comes at a computational cost, due to increasing parameter counts for storing world knowledge and growing context lengths that enable access to distant information, but incur a quadratic complexity penalty. Retrieval-augmented language modeling (RALM) alleviates this cost (Khandelwal et al., 2020; Yogatama et al., 2021; Borgeaud et al., 2022; Ram et al., 2023), as precise retrieval of relevant information can reduce memory and computation requirements. Moreover, RALM is beneficial for factuality, freshness and generalization without necessitating retraining, simply by swapping the retrieval index (Guu et al., 2020; Lewis et al., 2020; Huang et al., 2023).

However, past work on RALM has by and large *not* trained the retriever as a first-class component of the LM. In some cases (Khandelwal et al., 2020; Yogatama et al., 2021; Borgeaud et al., 2022), the retriever was used only at test time, or remained fixed throughout training, preventing it from adapting to the LM generator. In other cases, the retriever component was jointly trained but only after a separate pretraining phase for both the retriever and LM (Sachan et al., 2021; Izacard et al., 2022; Jiang et al., 2022; Bertsch et al., 2023). Thus, the retriever was not pre-trained from scratch with the LM, and only a fraction of the training budget was allocated for joint training.

Recently, Zhong et al. (2022) presented a retrieval-augmented LM that trains a retriever from scratch jointly with the LM, but (a) the retriever was trained to exploit *lexical* information only, and (b) the retrieved information was not fused at the *representation level* back into the LM.

In this work, we present the *Retrieval-Pretrained Transformer (RPT)*, a retrieval-augmented LM, where the retriever is a first-class component, trained jointly from scratch with the LM. RPT relies on two technical contributions. First, on the architecture side (see Fig. 1), input representations for the retriever are computed from the LM representations themselves (which we dub *self-retrieval*), and retrieved representations are fused back into the LM decoder for making next word predictions. Second, we train the retriever with an *auxiliary loss function* that encourages retrieving text fragments that increase the probability of generating the subsequent text. Specifically, given a recently-generated chunk c_t, the retriever is trained to retrieve chunks c_i that increase the probability of $p_{\text{scoring}}(c_{t+1} \mid c_i, c_t)$ according to a reference *scoring LM*. Fig. 1 provides an illustrative example for a case where a crime scene is described, and a scoring LM shows the benefit of retrieving a chunk thousands of tokens away (chunk 13) compared to lexical retrieval, which leads to a chunk that is only superficially related (chunk 100).

We focus on the problem of modeling long documents, such as books, articles, code, scripts, and dialogue, since these are naturally occurring examples of long-form content, where the entire index can be held within memory in a forward-pass. We evaluate RPT on four language modeling tasks and find that it improves perplexity across all tasks, outperforming prior work (Hutchins et al., 2022; Wu et al., 2022) as well as strong baselines (Borgeaud et al., 2022; Zhong et al., 2022). Moreover, we show that RPT retrieves high-quality chunks compared to retrievers that rely on lexical information. Based on our empirical findings, we argue RPT can pave the way toward the next generation of pre-trained LMs, where retrieval is strongly embedded within the architecture and training procedure.

2 Background

To situate our contribution, we review relevant recent RALM work. We extend this to more related work in §6.

Early work on RALMs, such as kNN-LM (Khandelwal et al., 2020) used retrieval to improve language modeling by interpolating the next-word distribution produced by the LM with a distribution proposed through a *test-time-only* retrieval mechanism. Borgeaud et al. (2022) later proposed Chunked Cross-Attention (CCA), where retrieval is performed also at training time, and retrieved representations are deeply fused into the representations produced by a Transformer decoder through attention. However, the retriever was trained separately and kept fixed during training, which prevented it from adapting to the LM over the course of training.

TRIME (Zhong et al., 2022), like this work, trained a retrieval-augmented LM from scratch where the retriever component and the decoder LM are trained jointly. Our work differs from TRIME in two aspects: First, TRIME, like kNN-LM, incorporates information from the retriever in a shallow manner through distribution interpolation, while we adopt CCA as a deeper fusion mechanism. Second, TRIME takes advantage of lexical clues for supervising the retriever, that is, given a query, the TRIME retriever learns to retrieve contexts that will lead to generating the same token as the query. We, on the other hand, use a scoring LM to evaluate what text chunks are relevant for increasing the probability of the chunk being generated, which leads to more semantic retrieval. This is similar to EPR (Rubin et al., 2022), which used this idea for learning to retrieve prompts for in-context learning, and perplexity distillation in Atlas (Izacard et al., 2022). However, Atlas does not train the retriever and LM from scratch and is an encoder-decoder model, more suitable for knowledge-intensive tasks. We, conversely, train from scratch and use a decoder model, more suitable for modeling long texts.

3 Retrieval-Pretrained Transformer

Problem Setup RPT, like RETRO (Borgeaud et al., 2022), is a chunk-wise retrieval-augmented LM, where the input sequence is divided into

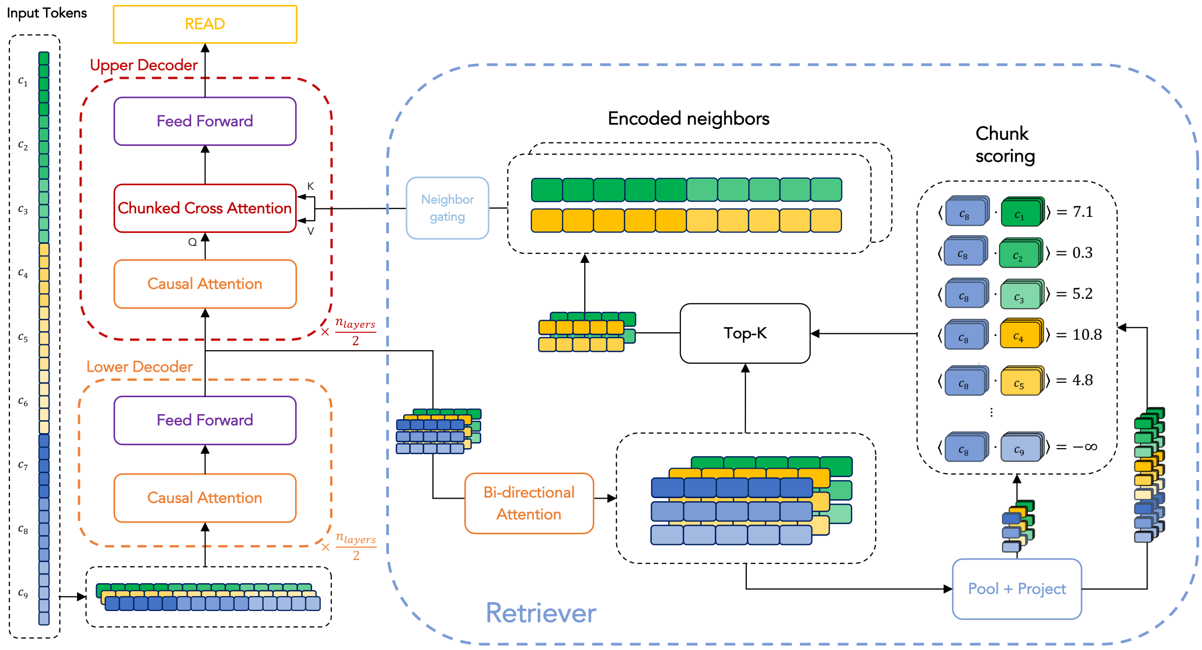

Figure 2: The architecture of the *Retrieval-Pretrained Transformer*, where an input of 45 tokens is shown, consisting of 9 chunks, and causal self-attention is applied over 15 tokens. The left side shows the decoder stack, where the bottom $\frac{n_{\text{layers}}}{2}$ are standard Transformer decoder layers, and the top $\frac{n_{\text{layers}}}{2}$ layers also include chunked cross-attention layers that fuse information from retrieved chunks. The right side shows the retriever, which takes a chunk and retrieves the highest-scoring K chunks that appeared earlier in the document.

chunks, and retrieval is performed at the chunk level. Specifically, given a sequence of L input tokens, $(x_1, x_2, \ldots, x_L)$, we partition it into a sequence of $\ell = \frac{L}{m}$ non-overlapping chunks of length m, denoted by $\mathcal{C} = (c_1, c_2, \ldots, c_\ell)$. For every possible *query* chunk, $c^{\mathbf{q}} = c_i$, the model will retrieve a subset of at most $K \ll \ell$ chunks, $\mathcal{R}(c^{\mathbf{q}}) \subset \mathcal{C}^{<i} = (c_1, c_2, ..., c_{i-w})$, where $\mathcal{C}^{<i}$ is the set of *retrievable* chunks for c_i, which excludes the w chunks to which it already has access to through causal self-attention. The goal is to learn a model that retrieves a chunk subset, $\mathcal{R}(c^{\mathbf{q}})$, that increase the probability of autoregressive generation of the *target chunk* $c^{\mathbf{t}} = c_{i+1}$.

We present our method in two parts. First, our architecture (§3.1), which leverages CCA to fuse retrieved representations into the LM, but adds a learned retriever component. Second, we present the training method (§3.2-§3.3), where the retriever is trained to retrieve chunks useful for generating a future chunk according to a reference LM.

3.1 Model Architecture

Fig. 2 illustrates our architecture, where the input has 45 input tokens divided into 9 chunks, and causal self-attention is applied over $w = 3$ chunks (15 tokens). The left side depicts the decoder stack (*"reader"*), and the right side the retriever. The reader is split into two, where the bottom $\frac{n_{\text{layers}}}{2}$ layers (*lower decoder*) are standard Transformer decoder layers that take w chunks as input and output representations that will be used by the retriever and the top decoder layers.

The top $\frac{n_{\text{layers}}}{2}$ layers (upper decoder) use Chunked Cross-Attention (CCA) to fuse information from the top-K neighbor chunks retrieved by the retriever back into the LM. We use standard CCA layers from RETRO (Borgeaud et al., 2022), where for each one of the ℓ chunks, queries are the m token representations of that chunk output by causal attention, and the keys and values are the token representations for the top-K neighbor chunks output by the retriever. For full details of CCA, see Borgeaud et al. (2022).

Next, we describe the retriever component, along with a neighbor gating mechanism for modulating the effect of retrieved representations.

Retriever The retriever takes as input the representations output by the lower decoder and produces a similarity score for every pair of chunks. Given a *query chunk* $c^{\mathbf{q}}$, the *query-based score* for each retrievable chunk c is $s_{\mathbf{q}}(c) = \langle W_Q \mathbf{c}^{\mathbf{q}}, W_K \mathbf{c} \rangle$, where $W_Q, W_K \in \mathbb{R}^{d \times d}$ are learned linear projections, and $\mathbf{c^q}$ and $\mathbf{c}$ are chunk representations.

For an m-token long chunk c, we compute its representation $\mathbf{c}$ by applying bidirectional attention over the chunk tokens, followed by mean-pooling across the time dimension. This maintains causality, as these representations are only used during the prediction of the next chunk.

Once scores for all pairs of chunks are computed, the *retrieved neighbor chunks* $\mathcal{R}(c^{\mathbf{q}})$, for each query chunk, $c^{\mathbf{q}}$, consists of its top-K highest-scoring retrievable chunks. Then, for each chunk $c_j \in \mathcal{R}(c^{\mathbf{q}})$, we concatenate the representations of the succeeding chunk c_{j+1} to provide additional context, and the final representation for all neighbors of all chunks is given by a tensor $C \in \mathbb{R}^{\ell \times K \times 2m \times d}$.[1]

Overall (and unlike methods like TRIME and kNN-LM), the retriever is an integral part of the LM, where the lower decoder computes representations for the retriever (which we dub *self-retrieval*), and the upper decoder consumes representations produced by the retriever.

Neighbor gating We add a neighbor gating mechanism to softly select neighbor representations that are useful for fusing into the upper decoder. Let $C_{i,k} \in \mathbb{R}^{2m \times d}$ be the token representations for the k'th neighbor of chunk c_i. We mean-pool across the time dimension to obtain a vector $\hat{\mathbf{c}}_{i,k}$ for each neighbor chunk. Then, we enrich the neighbor representation of each chunk by applying causal attention – a neighbor chunk representations $\hat{\mathbf{c}}_{i,k}$ attends to chunks that precede it or to neighbors of the same chunk c_i that are ranked higher. Finally, for each chunk we obtain the *gated retrieved representation* by multiplying the augmented representations by a gating score: $C^{\mathbf{g}}_{i,k} = \max\{\eta, \sigma(\frac{\mathbf{w}_{\text{ng}}\hat{\mathbf{c}}_{i,k}}{d})\} \cdot C_{i,k}$ where $\mathbf{w}_{\text{ng}}$ is a learned parameter vector, η is a small value meant to maintain gradient flow,[2] and σ is the sigmoid activation. Finally, in the upper decoder, when CCA is performed, the keys and values are $C^{\mathbf{g}}_{i,k}$.

3.2 Supervision Signal

For each query chunk $c^{\mathbf{q}} = c_i$, we want to identify neighbor chunks that will be helpful for generating $c^{\mathbf{t}} = c_{i+1}$, and use those neighbor chunks as supervision signal for the retriever. Similar to Rubin et al. (2022), we can exploit the fact that we are producing *training data* and use information from $c^{\mathbf{t}}$ itself to produce such a score. Unlike Zhong et al. (2022), who use lexical clues alone, we will use an independent *scoring LM* for this purpose.

Scoring every chunk w.r.t to all preceding chunks is quadratic in the number of chunks in a document, and thus computationally difficult. Thus, we use a simple, BM25 unsupervised retriever (Robertson and Zaragoza, 2009) that takes as input the concatenation of the chunks $(c^{\mathbf{q}}, c^{\mathbf{t}}) = (c_i, c_{i+1})$ and returns a set of candidates neighbor chunks, $\bar{\mathcal{R}} \subset \mathcal{C}(c^{\mathbf{q}})$, which have high lexical overlap with the current and subsequent chunk. This retriever has access to the tokens that need to be generated by the LM, which is allowed at training time.

Let $\hat{g}$ be an independently-trained LM, and let $\bar{c}_j$ be the concatenation (c_j, c_{j+1}). We compute a score $s_{\mathbf{t}}(\bar{c}_j)$ that reflects whether the information in $\bar{c}_j$ is more useful for decoding $c^{\mathbf{t}}$ compared to chunks that are close to $c^{\mathbf{q}}$. Specifically, the *target-based score* for a candidate chunk is

$$s_{\mathbf{t}}(\bar{c}_j) = \log \frac{\operatorname{Prob}_{\hat{g}}\left(c^{\mathbf{t}} \mid c_j, c_{j+1}, c^{\mathbf{q}}\right)}{\operatorname{Prob}_{\hat{g}}\left(c^{\mathbf{t}} \mid c_{i-2}, c_{i-1}, c^{\mathbf{q}}\right)}.$$

This score is positive when information in $\bar{c}_j$ is more useful for decoding $c^{\mathbf{t}}$ than information in the preceding two chunks (c_{i-2}, c_{i-1}).

We apply this scoring function to all chunks, and define for each query chunk $c^{\mathbf{q}}$ the set of *positive chunks* $\mathcal{R}^{\mathbf{q}}_{\text{pos}}$, which includes candidates for which $s_{\mathbf{t}}(\cdot) > 0$. This should result in helpful chunks, as each candidate chunk is at least as good as the local context. With this ordering at our disposal, we can apply standard retrieval training methods.

3.3 Training

To train the parameters of the retriever component, we adapt the widely-used LambdaRank loss (Burges et al., 2006). The loss for each query chunk $c^{\mathbf{q}}$ (w.r.t its retrievable chunks) is:

$$L_{\text{ret}}(c^{\mathbf{q}}) = \sum_{\{j,l:\bar{c}_l \in \mathcal{R}^{\mathbf{q}}_{\text{pos}}, s_{\mathbf{t}}(\bar{c}_l) > s_{\mathbf{t}}(\bar{c}_j)\}} \lambda_{jl} \max\left(0, \tau - (s_{\mathbf{q}}(c_l) - s_{\mathbf{q}}(c_j))\right)$$

where τ is a margin hyper-parameter, and λ_{jl} is the LambdaRank scaling that considers the relative ranking of each candidate. This loss is non-zero when for some pair of candidates, the target-based score disagrees (with margin τ) with the ranking of

[1]Similar to RETRO, token representations of retrieved chunks are also augmented through cross-attention over tokens of the query chunk, $c^{\mathbf{q}}$.

[2]We set $\eta = 0.1$ in all of our experiments.

the query-based score for candidates in $\mathcal{R}^{\mathbf{q}}_{\text{pos}}$. Optimizing this loss function allows RPT to distinguish between relevant and irrelevant chunks. Our final loss is $L_{\text{LM}} + \alpha_{\text{ret}} L_{\text{ret}}$, where L_{LM} is the standard LM loss and α_{ret} is the retrieval loss coefficient, increased linearly in the first 100K steps. We also increase τ linearly during training.

3.4 Important Implementation Details

Scheduled sampling To reduce train-test mismatch, we apply scheduled sampling (Bengio et al., 2015) during training. Namely, After computing the top-K neighbor chunks, we use these neighbors with probability $1 - p_{\text{ss}}$, and with probability p_{ss} the top-K scoring candidates from $\mathcal{R}^{\mathbf{q}}_{\text{pos}}$ as input for CCA. We anneal p_{ss} from 1 to 0 during the first 90% of training with a cosine schedule. This allows the model to gradually learn to use its own predictions. We report the effect of this in §5.3.

Sliding window attention at training and inference time As described in §3, the decoder takes as input w chunks, each with m tokens as input, and applies causal attention over them. In practice, to give the first tokens access to past tokens, we use the sliding-window attention mechanism (Dai et al., 2019; Beltagy et al., 2020; Hutchins et al., 2022), where the number of tokens in a window is 2,048 and the stride is 1,024. Thus, the input to each window is 2,048 tokens and the output are the representations for the last 1,024 tokens, which use the keys and values of the previous 1,024 tokens for contextualization.

At inference time a similar procedure is applied (Dai et al., 2019), where we compute and cache the key and value representations for segments of 1,024 tokens, and then use these as context for generating or estimating the probability of the next segment. Naturally, at inference time the retriever component provides access to all tokens from the beginning of the document.

Additional details At training time we use sequences of length $L = 16,384$ tokens, which are split into 4 devices, each consuming $4,096$ tokens. As mentioned, the decoder stack takes $2,048$ tokens as input (in a sliding window approach), which contains $\ell = 32$ chunks of length $m = 64$. We employ Rotary Positional embedding (Su et al., 2021), and train all models for 500K steps on a TPUv4-64, with an effective batch size of 2^{17} tokens.

Name	Tokens (Train/Test)	Median Length
ArXiv	12,000 / 16	16,368
CodeParrot	5,000 / 5	29,269
PG19	3,000 / 9	82,659
Books3	25,000 / 35	113,496

Table 1: Number of tokens (in millions) for each dataset and median document length.

For all models trained, we use the GPT-NeoX (Black et al., 2022) tokenizer, which was trained on the Pile (Gao et al., 2021a) and covers the domains we evaluate on (see §4). As our scoring language model, we use the deduplicated 1.4B parameter version of Pythia (Biderman et al., 2023), and score with it the top-20 BM25 candidates. Our model has 12 layers, hidden dimension $d = 1024$, and 8 attention heads with a head dimension of 128. We apply CCA every 2 layers and use 2 neighbors, unless mentioned otherwise. Additional implementation details are in Appendix A.1.

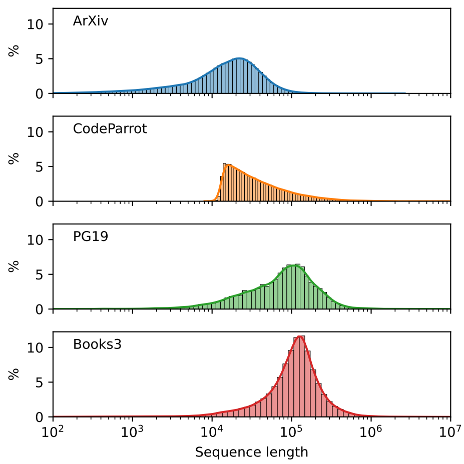

Figure 3: Histograms of the distribution over document length in tokens across all datasets. The x-axis is in log scale.

4 Long Range LM Datasets

We evaluate RPT on four datasets, covering domains such as books, code, and mathematical writing, which require the ability to recall information over long distances. Tab. 1 and Fig. 3 provide statistics on dataset size and the distribution over document length, showing that documents are long across all datasets and in particular PG19 and Books3, where documents typically contain 10^5 tokens or more. We briefly review the datasets.

PG19 Introduced in Rae et al. (2020), PG19 is a widely-used long-range language modeling benchmark containing books from Project Gutenberg, and covering a wide range of literary genres, styles, and topics. We adopt the exact setup and data split from prior work (Wu et al., 2022; Hutchins et al., 2022; Mehta et al., 2023).

Books3 is a corpus of books released as part of the Pile (Gao et al., 2021a), containing a vast collection of literary works from different domains. To our knowledge, we are the first to use this corpus as a long-range language modeling benchmark.

CodeParrot (Wolf et al., 2023) is a corpus of clean, nearly-deduplicated Python code from various GitHub repositories. Modeling code requires understanding patterns and contextualizing information over long distances, making it a natural candidate for testing long-range LMs. In our experiments, we follow the approach of Wu et al. (2022), combining files from the same repository to construct a corpus with longer sequences, and create a train/test split (see Tab. 1).

ArXiv is a corpus of preprint papers extracted from ArXiv. It consists of mathematical texts that require maintaining coherence and referring to previously mentioned information over extended text. Prior work evaluated long-range LMs on this corpus (Wu et al., 2022; Hutchins et al., 2022; Mehta et al., 2023), but did not release their corpus. Thus, we use the preprocessed corpus and data splits made available by Azerbayev et al. (2023).

5 Experiments

We now turn to experiments for comparing RPT to prior work across our four datasets.

5.1 Experimental Setup

We compare to the following baselines and oracles.

Transformer-XL Our simplest baseline is a standard transformer decoder stack with sliding window attention. Put differently, we simply remove from RPT the retriever component and CCA layers in the upper decoder. Using sliding window attention (as described in §3.4) can be viewed as a variant of Transformer-XL (Dai et al., 2019).

RETRO (Borgeaud et al., 2022) A retrieval-augmented model, where we omit the retriever component and feed the top-K neighbors retrieved by BM25[3] as input to the CCA layers in the upper decoder. During training, we use the query $(c^{\mathbf{q}}, c^{\mathbf{t}})$, since we have access to the target chunk. During inference, we use $c^{\mathbf{q}}$.

RPT-Lex A version of RPT, where the training signal is not obtained from the scoring LM, but from lexical information only, similar to TRIME (Zhong et al., 2022). Explicitly, the set of positive chunks $\mathcal{R}^{\mathbf{q}}_{\text{pos}}$ for a chunk $c^{\mathbf{q}}$ contains the top-20 chunks that have the highest BM25 score with $(c^{\mathbf{q}}, c^{\mathbf{t}})$.

RPT-Sem Our full model described in §3.

Block-Recurrent Transformer We use the official training implementation[4] of Block-Recurrent Transformer (Hutchins et al., 2022) with the default configuration.

Memorizing Transformer We use the official implementation[4] of Memorizing Transformers (Wu et al., 2022), with the default configuration and a memory size of 32K tokens.

Oracles For each test chunk, we can exhaustively search and use at test time the best possible neighbors for a model according to the scoring LM. This provides an upper bound for the performance of RPT-Lex and RPT-Sem, as they are trained to imitate the ranking produced by this oracle.

Metrics We use perplexity to evaluate the performance of models. In addition, we use the target score $s_{\mathbf{t}}(\cdot)$ from the scoring LM to compute for each chunk a gold ranking over all previous chunks, and to label chunks as positive/negative iff their target score is positive/negative, respectively. With this information, we can evaluate Precision@k, which is the fraction of top-k chunks according to the query-based score that are positive, and Recall@k, which is the fraction of positive chunks that are in the top-k chunks according to the query-based score. We also use the gold ranking to compute NDCG@k, which is a standard retrieval metric (Järvelin and Kekäläinen, 2002).

5.2 Results

Table 2 shows our main results, which show that RPT-Sem is comparable or better than all

[3]Concurrent work (Doostmohammadi et al., 2023) showed that training RETRO using BM25 substantially outperforms dense retrieval methods.

[4]https://github.com/google-research/meliad.

Model	**ArXiv**	**Code**	**PG19**	**Books3**	**Params**
Transformer-XL (ours)	3.11	2.30	11.48	15.00	202M
Retro w. BM25 (ours)	2.94	2.17	11.44	14.60	236M
RPT-Lex	2.92	2.23	11.59	14.32	242M
RPT-Sem	2.77	2.17	10.96	13.91	242M
w. 3 neighbours	2.75	2.16	**10.92**	**13.87**	242M
w. 4 neighbours	**2.74**	**2.15**	10.93	13.91	242M
Memorizing Transformer	2.92	2.18	10.97	14.40	212M
Block-Recurrent Transformer	2.89	2.73	10.95	14.64	212M
RPT-Lex w. Oracle	2.80	2.12	10.88	13.30	242M
RPT-Sem w. Oracle	2.69	2.10	10.26	12.74	242M

Table 2: Test set perplexity for all datasets. Unless specified, we use 2 neighbours during inference.

other baselines in all cases. Using a fixed retriever (RETRO) categorically improves performance compared to Transformer-XL; RPT-Lex leads to gains in Books3 but to losses in PG19 compared to RETRO, and RPT-Sem outperforms Transformer-XL, RETRO, and RPT-Lex on ArXiv, PG19, and Books3, and has performance comparable to RETRO on CodeParrot.

Compared to Block-Recurrent Transformers and Memorizing transformers, which do not use CCA, performance is again either comparable or better, with notable gains on ArXiv, CodeParrot, and Books3.

CCA allows one to dynamically increase the number of neighbors at inference time. When using 3 or 4 neighbors (instead of 2), performance improves, which allows one to trade compute for performance.

Last, oracle models consistently achieve the best perplexity across all datasets, improving from 2.74→2.69 on ArXiv, 2.15→2.10 on CodeParrot, 10.92→10.26 on PG19, and 13.87→12.74 for Books3. This shows that improving the training of the retriever can further improve performance.

Dataset	**Precision@2**			**Recall@10**			**nDCG@20**		
	BM25	RPT-L	RPT-S	BM25	RPT-L	RPT-S	BM25	RPT-L	RPT-S
ArXiv	27%	26%	32%	55%	54%	58%	24%	24%	30%
Code	29%	26%	34%	53%	52%	56%	25%	23%	30%
Books3	23%	19%	26%	55%	50%	58%	18%	16%	22%
PG19	22%	22%	28%	55%	55%	61%	18%	18%	23%

Table 3: Test retrieval metrics across datasets.

Retrieval metrics Table 3 presents the retrieval metrics w.r.t oracle positive chunks. Again, retrieval with RPT-Sem outperforms both RPT-Lex and BM25 in all cases. This shows the importance of training a retriever, and moreover that using semantic supervision leads to better retrieval compared to a lexical signal only.

Model	**ArXiv**	**Code**	**PG19**	**Books3**
RPT-Sem	2.77	2.17	10.96	13.91
- Only Teacher forcing	2.91	2.22	11.54	14.66
- No Teacher forcing	2.95	2.26	13.10	14.40
- No Neighbor Gating	2.92	2.20	11.50	18.68

Table 4: Results of our ablation study on RPT-Sem.

Distribution of improvements across chunks We compute the improvement in perplexity for all chunks when comparing to Transformer-XL and plot the distribution of improvements for RETRO, RPT-Lex, and RPT-Sem in Fig. 4. Clearly, RPT-Sem has a heavier right tail in all cases except for CodeParrot, further illustrating its advantage over the other baselines. We further analyze why RETRO with BM25 performs well on CodeParrot in §5.4.

5.3 Ablations

Tab. 4 shows the result of an ablation study on RPT-Sem over all datasets.

Only Teacher Forcing We force the model to attend to gold neighbors according to the scoring LM, without annealing p_{ss} during training. This leads to a performance drop across all datasets, and in particular for PG19 and Books3.

No Teacher Forcing Here, we do the opposite and fix $p_{\text{ss}} = 0$ throughout training, i.e., we only use the predicted neighbors and not gold ones. This can lead to undertraining of the CCA layers since they are exposed to low-quality neighbors at the beginning of training and results drop even further compared to Only Teacher Forcing.

No neighbor gating We disable neighbor gating which controls the flow of information from neighbor chunks and analyze the effect on model performance. We observe a performance reduction across all datasets, notably on Books3, where

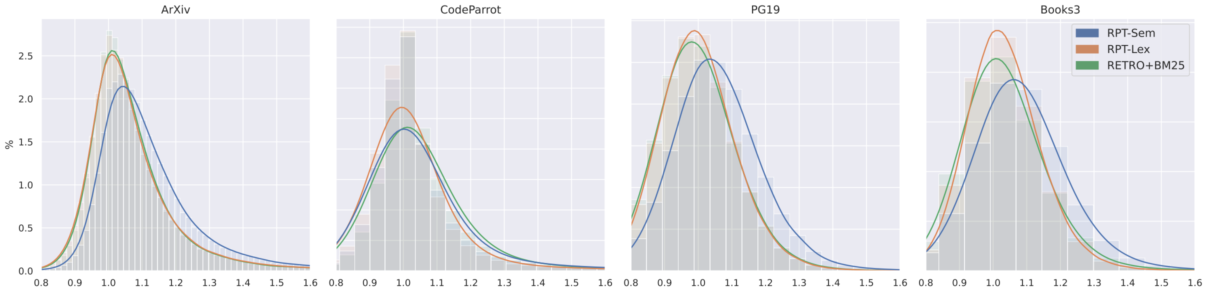

Figure 4: Relative perplexity improvement across different retrievers. All retrievers exhibit positive skew with a heavy right tail, and RPT-Sem leads to the most pronounced improvements.

perplexity increases by 4.5 points. Since neighbor gating is independent of the retriever used, we show results when adding neighbor gating to RETRO in §A.4., which shows mixed results.

5.4 Analysis

Figure 5: We measure the number of unique token overlap between query/target chunks and the best retrieved neighbor.

Token overlap Fig. 5 plots the average number of tokens that overlap between the query/target chunks the best retrieved neighbor for RETRO, RPT-Lex, and RPT-Sem. RPT-Sem retrieves paragraphs with higher overlap with the *target* chunk compared to RPT-Lex. Naturally, BM25 retrieves chunks with the highest overlap with the *query* chunk. However, this does not translate to higher lexical overlap for the *target* chunk.

Supervision quality We train RPT-Sem using information from the target scoring function $s_{\mathbf{t}}(\cdot)$,

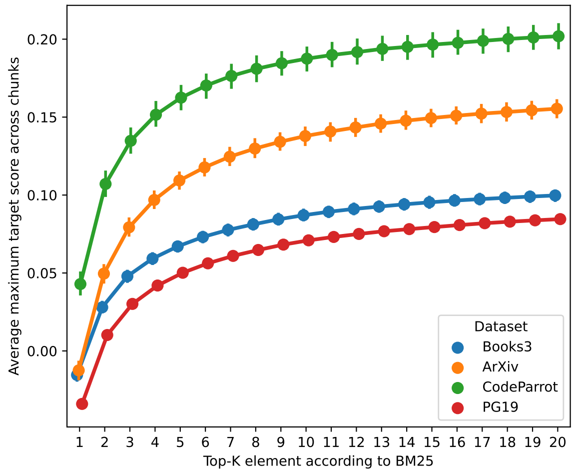

Figure 6: The maximal target score $s_{\mathbf{t}}(\cdot)$ for the top-k chunks retrieved by BM25 averaged across chunks and for all datasets. Since the maximal target score for the top-20 chunks is much higher than for the top-2, learning to rerank the top-20 BM25 candidates can lead to substantial improvements in retrieval quality.

which we saw leads to model improvements. However, the target scoring function only provides a reranking of the top-20 candidates according to BM25. Thus, a natural question is how much does the supervision quality improve through this reranking. Figure 6 shows for every rank k the maximal target score among the top-k chunks according to BM25, averaged over chunks and across our 4 datasets. Clearly, reranking the top-20 BM25 candidates has a lot of potential, as the maximal target score is much higher for the top-20 candidates compared to the top-2. This hints that longer and better training of the retriever can further improve the performance of RPT-Sem.

Interestingly, our analysis sheds light on why RPT-Sem outperforms RETRO clearly on Books3 and PG19 but less so on CodeParrot. The maximal target score for CodeParrot when $k = 2$ is already quite high – around 0.1, which corresponds to more than 10% improvement in the probability of the target chunk compared to the local context. Conversely, for PG19 and Books3, the target score when $k = 2$ is closer to 0. This hints that lexical information alone is quite effective for CodeParrot, potentially by retrieving function definitions, variable assignments, etc.

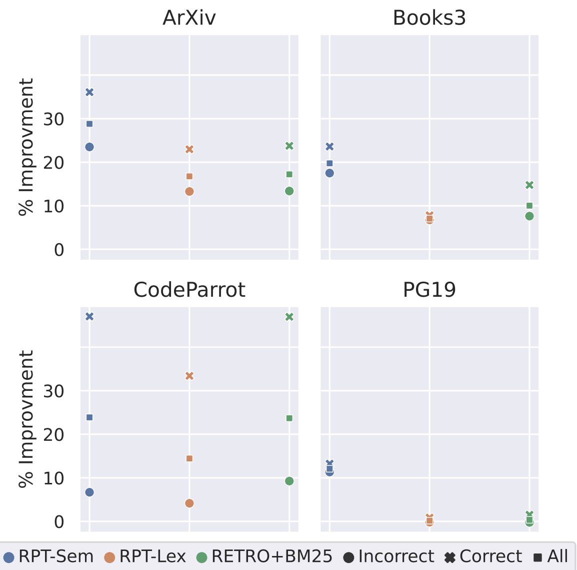

Figure 7: Relative improvement with/without correct retrieval.

Subgroup analysis Figure 7 shows the average relative improvement (across chunks) of RETRO, RPT-Lex, and RPT-Sem compared to Transformer-XL, when distinguishing between cases where a "gold" oracle chunk was retrieved and cases where no gold chunk was retrieved.

As expected, RPT-Sem leads to improvements on all datasets, and outperforms other baselines except for RETRO on CodeParrot where performance is similar. Second, cases where a gold chunk was retrieved indeed typically lead to larger improvements, but we witness improvements even in cases where a gold chunk was not retrieved, which shows that the model can still benefit from such retrievals.

6 Related Work and Discussion

Long-range language modeling A primary focus in long-range language modeling has been addressing the quadratic complexity of attention in order to develop more efficient mechanisms for handling long texts. For instance, Transformer-XL (Dai et al., 2019) processes the input using a segment-level mechanism while retaining a cache from previous segments. Longformer (Beltagy et al., 2020) extends this idea to accommodate even longer contexts. Sparse strategies, such as those proposed in Zaheer et al. (2020); Roy et al. (2021); Kitaev et al. (2020), attend to only a subset of tokens through clustering or hashing methods. Another approach involves compressing the input and attending over the compressed sequence (Martins et al., 2022; Rae et al., 2020), or learning to ignore irrelevant tokens (Sukhbaatar et al., 2021). Recently, recurrent mechanisms have re-emerged as potential solutions (Fan et al., 2021; Hutchins et al., 2022; Mehta et al., 2023). From an analysis perspective, past work (Press et al., 2021) demonstrated that standard LM benchmarks are not ideal for measuring the long-range capabilities of models. Sun et al. (2021) discuss various types of sequences that benefit from having a long context, and Rae and Razavi (2020) investigate long-range architectural choices and recommend increasing long-range capabilities in the upper layers.

Retrieval augmented LMs Retrieval-augmented LMs have emerged as a prominent approach for efficiently leveraging external knowledge while generating text. These models can be broadly divided into those operating at token-level granularity and those operating at sequence-level granularity. Token-level methods, such as kNN-LM (Khandelwal et al., 2020), TRIME (Zhong et al., 2022), and SPALM (Yogatama et al., 2021), retrieve information for individual tokens. Sequence-level approaches like RAG (Lewis et al., 2020) utilize pre-trained encoder-decoder models with pre-trained retrievers for tasks like open-domain question answering. Similarly, FiD (Izacard and Grave, 2021b) employs generative encoder-decoder models that fuse evidence from multiple passages during the decoding process, closely related to the CCA mechanism (see additional discussion in App A.3). Recently, Wang et al. (2023) demonstrated the potential benefits of conducting retrieval and chunked cross-attention at each time step, compared with the original RETRO (Borgeaud et al., 2022) paper, which retrieves every $m = 64$ steps.

Joint retriever-reader training Joint training approaches typically concentrate on transferring information between a pre-trained reader into a pre-trained retriever. These methods commonly involve updating the retriever index during the training process in the context of knowledge-intensive tasks, such as open-domain question answering. For instance, REALM (Guu et al., 2020) utilizes masked language modeling as a learning signal to update the retriever. EMDR2 (Sachan et al., 2021) extends FiD by using encoder-decoder models to back-propagate errors from the predicted answer to the retriever. Similarly, Izacard and Grave (2021a) demonstrate that it is possible to use attention scores from the reader to supervise the retriever. Notably, Izacard et al. (2022) further scale up these approaches and jointly train a retriever with an encoder-decoder model, demonstrating strong few-shot learning capabilities. They also investigate various retriever updating techniques to address train-test mismatches in the retrieval process. We do not encounter the issue of index update since we compute the entire index through a forward pass.

Attention as Retrieval Several works view the attention layer as a retrieval component. Memorizing Transformers (Wu et al., 2022) employ a single k-NN layer and retrieve cached keys and values without back-propagating gradients through the retrieval operation. Similarly, Bertsch et al. (2023) demonstrate that this approach can be used with any existing pre-trained model and apply it at every attention layer for long summarization tasks. Notably, Jiang et al. (2022) use this observation and employ a caching mechanism (Gao et al., 2021b) to enable joint end-to-end training with the supervision of the downstream task. We view the latter as a potential way to fine-tune RPT and leave it for future work.

Retriever Pre-training Early work on retriever pre-training relied on the unsupervised Inverse Cloze Task to pre-train the retriever (Lee et al., 2019; Guu et al., 2020). It was later shown that directly using BERT (Devlin et al., 2019) with a supervised objective is sufficient to get good performance on standard benchmarks (Karpukhin et al., 2020). However, this paradigm showed lackluster performance on long-tail entities compared to BM25 (Amouyal et al., 2022; Sciavolino et al., 2021). Recently, unsupervised pre-training methods (Gao and Callan, 2022; Ram et al., 2022; Izacard et al., 2021) enabled improved performance. However, these methods are initialized from a pre-trained BERT (Devlin et al., 2019) encoder model, while RPT is a retriever-reader architecture trained from scratch that outperforms BM25 without any additional pre-training.

Supervising retrievers with LLMs EPR (Rubin et al., 2022) demonstrated that LLMs could be employed to train a retriever for prompt retrieval by estimating the probability of an output given the input and a candidate training example as the prompt. Similar techniques were applied to open-domain question answering via re-ranking retrieval results (Sachan et al., 2022; Ram et al., 2023) and to supervise retrievers through perplexity distillation (Izacard et al., 2022). Recently, Shi et al. (2023) utilized this supervision method to improve the performance of various LLMs in a black-box fashion.

7 Conclusion

In this work, we present the Retrieval-Pretrained Transformer (RPT), a retrieval-augmented LM where the retriever is trained as a native component of the LM to retrieve semantically relevant chunks for future text prediction. We evaluate RPT on four long-range language modeling tasks, including books, code, and mathematical writing. We demonstrate that by seamlessly integrating the retriever into the architecture and training process, RPT benefits from the fusion of retrieved context, improving over strong retrieval-augmented baselines. We envision RPT will pave the way for a new generation of pretrained language models with retrieval deeply integrated throughout their architecture and training process.

Acknowledgments

This research was supported with Cloud TPUs from Google's TPU Research Cloud (TRC) and The European Research Council (ERC) under the European Union Horizons 2020 research and innovation programme (grant ERC DELPHI 802800). Ohad would like to thank Iz Beltagy for suggesting the TRC program, and the entire TAU NLP lab and especially Guy Dar and Itay Itzhak. This work was completed in partial fulfillment of the Ph.D. degree of Ohad Rubin.

References

Samuel Joseph Amouyal, Tomer Wolfson, Ohad Rubin, Ori Yoran, Jonathan Herzig, and Jonathan Berant. 2022. Qampari: An open-domain question answering benchmark for questions with many answers from multiple paragraphs.

Zhangir Azerbayev, Edward Ayers, and Bartosz Piotrowski. 2023. Proof-Pile: A Pre-training Dataset of Mathematical Text. `https://huggingface.co/datasets/hoskinson-center/proof-pile`.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv:2004.05150*.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1171–1179, Cambridge, MA, USA. MIT Press.

Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew R. Gormley. 2023. Unlimiformer: Long-range transformers with unlimited length input.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. 2023. Pythia: A suite for analyzing large language models across training and scaling.

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

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. 2022. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 2206–2240. PMLR.

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. 2020. Language models are few-shot learners.

Christopher Burges, Robert Ragno, and Quoc Le. 2006. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems*, volume 19. MIT Press.

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. 2022. Palm: Scaling language modeling with pathways.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota.

Ehsan Doostmohammadi, Tobias Norlund, Marco Kuhlmann, and Richard Johansson. 2023. Surface-based retrieval reduces perplexity of retrieval-augmented language models.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. 2021. Addressing some limitations of transformers with feedback memory.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2021a. The pile: An 800gb dataset of diverse text for language modeling. *ArXiv preprint*, abs/2101.00027.

Luyu Gao and Jamie Callan. 2022. Unsupervised corpus aware language model pre-training for dense passage retrieval. 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)*. Association for Computational Linguistics.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

Yangsibo Huang, Daogao Liu, Zexuan Zhong, Weijia Shi, and Yin Tat Lee. 2023. $k$nn-adapter: Efficient domain adaptation for black-box language models.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. 2022. Block-recurrent transformers. In *Advances in Neural Information Processing Systems*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *Trans. Mach. Learn. Res.*, 2022.

Gautier Izacard and Edouard Grave. 2021a. Distilling knowledge from reader to retriever for question answering. In *International Conference on Learning Representations*.

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*, pages 874–880, Online. 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. Atlas: Few-shot learning with retrieval augmented language models.

Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*.

Zhengbao Jiang, Luyu Gao, Zhiruo Wang, Jun Araki, Haibo Ding, Jamie Callan, and Graham Neubig. 2022. Retrieval as attention: End-to-end learning of retrieval and reading within a single transformer. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2336–2349, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick 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)*, pages 6769–6781, Online. 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.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of ICLR*.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *International Conference on Learning Representations*.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Patrick 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*, volume 33, pages 9459–9474. Curran Associates, Inc.

Pedro Henrique Martins, Zita Marinho, and Andre Martins. 2022. ∞-former: Infinite memory transformer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Dublin, Ireland. Association for Computational Linguistics.

Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. 2023. Long range language modeling via gated state spaces. In *The Eleventh International Conference on Learning Representations*.

Ofir Press, Noah A. Smith, and Mike Lewis. 2021. Shortformer: Better language modeling using shorter inputs. 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)*.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain. Association for Computational Linguistics.

Jack Rae and Ali Razavi. 2020. Do transformers need deep long-range memory? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7524–7529, Online. Association for Computational Linguistics.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models.

Ori Ram, Gal Shachaf, Omer Levy, Jonathan Berant, and Amir Globerson. 2022. Learning to retrieve passages without supervision. Association for Computational Linguistics.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. Learning to retrieve prompts for in-context learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2671, Seattle, United States. Association for Computational Linguistics.

Devendra Singh Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen tau Yih, Joëlle Pineau, and Luke Zettlemoyer. 2022. Improving passage retrieval with zero-shot question generation. In *Conference on Empirical Methods in Natural Language Processing*.

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*.

Christopher Sciavolino, Zexuan Zhong, Jinhyuk Lee, and Danqi Chen. 2021. Simple entity-centric questions challenge dense retrievers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen tau Yih. 2023. Replug: Retrieval-augmented black-box language models.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding.

Sainbayar Sukhbaatar, Da JU, Spencer Poff, Stephen Roller, Arthur Szlam, Jason E Weston, and Angela Fan. 2021. Not all memories are created equal: Learning to expire.

Simeng Sun, Kalpesh Krishna, Andrew Mattarella-Micke, and Mohit Iyyer. 2021. Do long-range language models actually use long-range context? *ArXiv*, abs/2109.09115.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Boxin Wang, Wei Ping, Peng Xu, Lawrence McAfee, Zihan Liu, Mohammad Shoeybi, Yi Dong, Oleksii Kuchaiev, Bo Li, Chaowei Xiao, Anima Anandkumar, and Bryan Catanzaro. 2023. Shall we pretrain autoregressive language models with retrieval? a comprehensive study.

Thomas Wolf, Loubna Ben Allal, Leandro von Werra, Li Jia, and Armel Zebaze. 2023. A dataset of python files from github. `https://github.com/huggingface/blog/blob/main/codeparrot.md` version=codeparrot/codeparrot-train-v2-near-dedup.

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.

Dani Yogatama, Cyprien de Masson d'Autume, and Lingpeng Kong. 2021. Adaptive semiparametric language models. *Transactions of the Association for Computational Linguistics*, 9:362–373.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big bird: Transformers for longer sequences. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models. *ArXiv*, abs/2205.01068.

Zexuan Zhong, Tao Lei, and Danqi Chen. 2022. Training language models with memory augmentation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5657–5673, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. 2020. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Conference on Neural Information Processing Systems*.

A Appendix

A.1 Additional Implementation Details

All models are implemented in JAX, we use a dropout rate of 0.05, weight decay of 1e-8, Cosine decay to 0.1 of the maximum learning rate, global gradient norm clipping of 1, and tied input embedding (Press and Wolf, 2017). For our optimizer we used AdaBelief (Zhuang et al., 2020), which is a version of Adam (Kingma and Ba, 2015) that instead of the accumulating squared gradients, accumulates the squared difference between the gradient and the momentum. In initial experiments, we found AdaBelief to increase stability. Similar to Block-Recurrent we found that lowering the learning rate was necessary for convergence while training on Code, so for CodeParrot, we lower the learning rate. For each dataset, we perform a grid search w.r.t τ, and set $\tau = 128$ for Books3, $\tau = 4$ for PG19, $\tau = 2$ for CodeParrot, and $\tau = 8$ for ArXiv. We set $\alpha_{\text{ret}} = 1e - 9$ for all datasets. Our base learning rate is $5e - 3$, and besides what is mentioned above, we do not tune other hyperparameters. We use the validation set to choose hyperparameters.

A.2 Scoring LM

We use the deduplicated 1.4B parameter version of the Pythia (Biderman et al., 2023) LM. We also performed early experiments with the T5 tokenizer and T5-XL 1.1, but since it was not trained on code or latex, Pythia 1.4B was preferable, since it was trained on the Pile.

A.3 Comparing to FiD

RPT shares similarities with Fusion-in-Decoder (FiD) (Izacard and Grave, 2021b). Both RPT and FiD employ cross-attention mechanisms to integrate the retrieved context within their models. In FiD, an initial retrieval is conducted, followed by encoding the retrieved neighbors separately, and finally integrating them into the model using cross-attention in the decoder. In RPT, the decoder computes chunk embeddings and performs native retrieval, and then chunked cross-attention is applied to fuse the retrieved context with the model's predictions. RPT also performs repeated retrieval at the chunk level throughout the generation process, rather than retrieving only once based on the initial prompt. This enables RPT to continually adapt and incorporate relevant information from prior chunks to generate subsequent tokens more effectively. Furthermore, RPT is trained with retrieval being an integral part of the model during the entire pre-training phase, in contrast with FiD which plugs in retrieval components to solve specific downstream tasks. We view RPT as more suitable for long-text generation tasks.

Model	ArXiv	Code	PG19	Books3
RETRO W. BM25 (OURS)	2.94	2.17	11.44	14.60
W. GATING	2.97	2.21	11.84	13.92
RPT-SEM	2.77	2.17	10.96	13.91

Table 5: Results of our ablation study w. neighbor gating.

A.4 RETRO with Neighbor Gating

Neighbor gating is a mechanism that can be applied to any retrieval-augmented LM, whether the retriever is trained or not. In Tab. 5, we show results of RETRO when adding neighbor gating. Results improve substantially on Books3, but deteriorate on PG19, and are roughly equivalent for ArXiv and CodeParrot.