A POOR MAN'S RETRIEVER FOR QA SYSTEMS:
NEW APPROACHES FOR COST EFFICIENT INFORMATION RETRIEVAL

Andrew Amore & Jose Pliego San Martin
Duke University
{andrew.amore, jose.pliego.san.martin}@duke.edu

ABSTRACT

Modern Question Answering (QA) systems consist of two components: readers and retrievers. Retrievers reduce the passage search space for answer extraction and limit the overall accuracy of QA methods. Conventional retrievers consume large amounts of resources, reducing their viability to large corporations or well funded institutions. In addition, some retrieval methods are prone to overfitting, requiring an expensive retraining process to understand new document sources. In this paper, we outline a methodology for building information retrieval systems on a limited budget and perform feature enhancement using transfer learning. Through several ablation studies we demonstrate that existing DPR approaches are very sensitive to small changes in the problem domain, and introduce an approach to potentially improve generalizability which outperforms the existing DPR framework under one ablation. We also highlight a potential data quality issue from a well-cited paper, which may call into question published accuracy metrics and warrant additional review. Code for this analysis can be found on GitHub.

1 INTRODUCTION

The goal of a question answering (QA) system is to produce accurate answers to a large variety of input questions. Applications of QA are vast, from the generalizable Google search engine, to more conventional customer service applications. General QA works by generating a probable answer from a set of documents, called a corpus, in response to a question. Questions can be unknown in advance and range in content and context from "*Who directed the movie Forrest Gump?*" to "*What is Islam?*", making this task challenging. Modern approaches consist of two components: readers and retrievers. The first-stage retrieval process limits overall QA accuracy, as it filters information for a later, more effective reader search. Because of this dependence, retrievers are vitally important to overall QA systems and are the focus of this paper.

In section 2, we briefly review modern and classical approaches to information retrieval (IR) and discuss its main challenges. In section 3, we introduce our "novel", cost-effective augmentation method, using a large-language-model (LLM) to transform initial document passages for more effective retriever search. Section 4 includes more detail on our experiments and highlights performance differences over a non-augmented baseline. In section 5, we analyze the results and comment on observed differences after tweaking some components of the retriever pipeline. Finally, section 6 contains a brief conclusion and reflection on the relevance of this work.

2 RELATED WORK

2.1 CLASSICAL RETRIEVAL METHODS

Prior to neural networks, the Probabilistic Relevance Framework (PRF), reviewed by Robertson & Zaragoza (2009), developed many successful QA algorithms, like BM25, which compares question and document passages by evaluating similar word frequencies. Passages with large amounts of matching words, identical to the input question, are more likely to be retrieved. A benefit of

PRF methods is they require no training and generalize well. However, they rely on exact word matches between textual sources, which can fail to recognize synonyms. Many retrieval methods are compared to BM25 as a performance benchmark.

2.2 Modern Retrieval Methods

Most modern approaches to QA facilitate answer generation using a two-stage reader/retriever architecture previously introduced. Transformer based readers, like BERT (Devlin et al., 2018), utilize a neural network to perform a thorough answer search across document passages and have demonstrated tremendous improvements over classical methods. Yang et al. (2019) were among the first to demonstrate the benefits of a two-component system, sharing BM25 retrieval results with a BERT based reader for QA, however, their error analysis revealed the overall performance is constrained by the initial BM25 retrieval. Adapting transformers to IR has been explored, however, these networks have a memory limitation, from self-attention, which restricts the length of input passages (Das et al., 2021) and makes an exhaustive search time-consuming. To address BM25 shortcomings, many modern approaches use a distance metric, like cosine similarity, to perform an aggregated comparison between continuous representations of textual data to identify relevant passages. The summary metric establishes a passage ranking and identifies the top-k most relevant passages to share with a reader. We now introduce a few modern approaches.

Embedding Approaches. Classical and modern approaches diverge in the representation of document corpora. Karpukhin et al. (2020) develop dense passage retrieval (DPR), which computes passage relevance using vector representations derived from neural embeddings. DPR optimizes these embeddings to maximize the inner-product distance between a question and the labeled answer passage. Significant performance gains were reported over a BM25 baseline, however, labeled data is required to fine-tune embeddings and training relies on a sensitive batching strategy to compute gradients, restricting the approach to small datasets (Izacard & Grave, 2020).

An alternative to passage based DPR, but also leveraging dense embeddings, is presented by Lewis et al. (2021). Their method compares new queries against an augmented query set developed from passages in the training data, which avoids the direct use of indexed passages. This approach greatly reduces the number of embedding parameters, as generated queries are significantly shorter in length than document passages, making it more memory efficient. Additionally, the augmented queries undergo a filtering process to remove wrong or ambiguous entries and provides a reported metric that can be used to assess the uncertainty in retrieved passages. Our proposed generation method is very similar to Lewis et al. (2021), however, we utilize trained DPR embeddings from Karpukhin et al. (2020) and append our generated questions to original passages for retrieval evaluation.

Closed-Book QA. More recently, there is growing literature around unsupervised QA systems which disregard the reader/retriever architecture in favor of a single seq2seq model utilizing an encoder-decoder architecture. These approaches avoid the use of labeled datasets and enable a greater number of training examples. Roberts et al. (2020) showed these models can achieve state-of-the-art results, but are prohibitively large with billions of parameters that effectively "memorize" question answers. A timely example is ChatGPT, developed from the GPT-3 model, and demonstrates an astounding ability to generate complex answers from user posed questions. To get a sense of the unfathomable number of parameters, GPT-3 has **170,000,000,000** and the T5 model used in our analysis has "only" 220 million! There are substantial costs, not only train, but simply to use these models. In our experience, running just 30,000 passages through an inference pipeline can occupy a single, energy-hungry, GPU for multiple days. Production models are often deployed across parallel GPU architectures to speed up inference, but the energy use will be the same. Resource usage (i.e. electricity) is an overlooked, but critical component to consider when deploying these systems.

Reranking. An even newer approach to IR combines the benefits of classical and modern methods using two-stage retrieval. Sachan et al. (2022a) introduce a second-stage reranking algorithm that can be applied to existing retrieval methods, like BM25. During reranking, a LLM computes conditional likelihood estimates of an input query, given each initially retrieved passage, and reorders them corresponding to the new likelihood estimates. This procedure displays state-of-the-art results on full open domain QA, however, reordering large numbers of passages increases latency, as relevance scoring is costly, and the full pipeline still depends on the initial retrieval process. Our work involving a LLM was inspired by this analysis.

3 Approach

3.1 Generalization

QA systems receive questions in a variety of formats (yes/no, short-answer, etc.) which can contribute to poor generalizability of DPR systems. Additionally, DPR methods utilize a batching based training routine that requires a negative passage set, called in-batch negatives, for gradient computation. However, the majority of passages within a training corpus will be irrelevant, containing information about vastly different subjects, and the variety of in-batch negative sets can alter the update procedure of the network and dramatically alter the learned associations.

To address generalizability, we propose a data augmentation strategy for IR leveraging a LLM similar to Lewis et al. (2021). We rationalize that each passage contains only enough knowledge to answer a latent, but fixed set of questions, we define as a question answering capacity. If we can uncover this latent factor, and directly provide it as an input, we may allow the model to infer more general knowledge representations from an augmented feature set. In our method, before a passage is included in the retrieval corpus, it undergoes feature enhancement to generate a set of candidate questions relevant to that passage from a generative model fine-tuned on the Google T5-Base. This newly created question set is appended to the original passage for inner-product maximization during retrieval. To test the effectiveness of generated questions, we concatenate 1, 2, and 3 consecutive passages together within the same document and define it as the concatenation level. For the vast majority of the passages, appending the questions does not exceed the maximum input length of the encoders. For the few that exceed this limit, we truncate at 512 tokens.

We hypothesize several potential benefits of this approach that may increase the viability of DPR methods on larger corpora. If the generative set is representative of the question answering capacity, we exhaust the answering potential of each passage, allowing for more informed comparisons using the passage augmented set. Second, by including text formatted as questions we allow the model to compare information in similar formats: question to passage+question instead of question to passage like original DPR. Lastly, generated questions may provide a "memory-bank" of commonly asked questions, similar to an FAQ page, that could provide a roadmap for a model to develop "shortcuts" linking input questions to augmented variations.

3.2 Resource Intensive

To curate a labeled QA dataset, a manual annotator, posed with a question, must identify the correct answer passage from an internet archive, often Wikipedia. This process is time-consuming, error-prone, and can limit the number of training examples which encourages overfitting. In addition, both classical and modern methods utilize an in-memory document index to facilitate efficient passage retrieval, as the sheer number of comparisons make computations either memory or time intensive. These indexes use substantial amounts of expensive RAM with corpora that can number in the billions. For example, the 2018 Wikipedia corpus from this analysis consists of over 20 million passages and requires over 5TBs of memory to index at once, which can cost upwards of $20,000.

To address challenges, we elect to use the trained DPR retriever[1] developed by Karpukhin et al. (2020) with our augmented passages in an attempt to minimize the dependence on labeled QA data. Resource constraints were a big factor in our analysis. In our experiments, feeding the tokenized passages through the BERT encoder was a significant computational bottleneck and our limited budget made it impractical to store both encoder parameters and dense representations of the full corpus. To address resource limitations, we take a random sample[2] to reduce the number of passages and work around memory constraints during the encoding stage by feeding passages through the encoder in batches of 10. We then save dense representations out-of-memory and later reload them without the encoder parameters to perform indexing which reduces memory consumption. While taking a random sample prohibits our ability to draw conclusions from the full corpora, we believe the sample size is large enough to warrant consideration for larger scale experiments.

[1] Rationale is discussed in 4.2

[2] Discussed in more detail in 4.1.2

4 EXPERIMENTS

We introduce the data used for experiments, highlight data quality concerns and discuss basic model design. We focus our analysis to a subset of the final retrieval corpus to alleviate resource constraints.

4.1 DATASETS AND EVALUATION[3]

Wikipedia. To represent a retrieval corpus, we download the English Wikipedia dump of December 2018 from the official DPR repository, emulating a number of cited work[4]. The dataset archives all published Wikipedia articles (documents) from the time period and consists of three fields: id (unique integer), title, and passage. Each document is divided into a series of 100-word passages and includes the document title prepended to the first passage. Documents can be reconstructed in the correct order using the unique id. The raw data was uploaded to a BigQuery database for modeling.

Natural Questions (NQ). For retrieval evaluation, we include labeled question/answer pairs from the Google NQ benchmark (Kwiatkowski et al., 2019). It consists of real queries, posed by users to the Google search engine, and corresponding Wikipedia answer passages. The document information provided by Google is raw HTML and would require extensive data cleaning for language modeling. The BEIR Benchmark (Thakur et al., 2021) provides a parsed version of the original dataset, which we downloaded from the official mirror. BEIR includes the original queries, a subset of Wikipedia articles used for answer identification, and lookup tables linking queries to corresponding answer passages. The format of Wikipedia documents follows a similar convention to the DPR dataset, including a unique passage identifier, document title and segmented passage entries. Raw files from both NQ-Test and NQ-Train were uploaded to BigQuery for evaluation.

4.1.1 DATA QUALITY ISSUES

Duplicate Data. Both Wikipedia document sources have duplicate passages, defined as observations with distinct identifiers, but identical titles and text contents. Passages may repeat throughout a document, however, given the considerable passage length, is unlikely. Table 1 presents a summary by datasource[5] and reveals NQ-Train is substantially more "duplicated" than other sources.

Data	Initial Passages	Distinct Passages	Percent Duplicated
Wikipedia DPR	21,015,324	20,975,394	$< 1\%$
NQ-Train (BEIR)	18,060,996	7,332,438	59%
NQ-Test (BEIR)	2,681,468	2,670,337	$< 1\%$

Table 1: Distinct and total record counts by data source.

Failing to remove duplicate entries can effect retriever accuracy and increase memory usage. To address duplication, we assume it is the result of a processing routine which inaccurately includes additional passage copies at the end of each document and opt to keep passage copies with the lowest unique identifier. We perform the removal process at different concatenation levels, to avoid removing consequent passages[6], as we suspect a more random duplication pattern.

Missing Wikipedia Answer Passages. The DPR Wikipedia corpus consists of over 3 million documents, but, when compared to Wikipedia data in BEIR, appears to exclude over 10% of answer documents based on a distinct title comparison. Table 2 shows the number of document articles in each BEIR corpus and displays a substantial number of missing DPR entries in both BEIR datasets[7]. It's plausible some Wikipedia articles in the NQ dataset may not have existed in December 2018, when the DPR data was harvested. Kwiatkowski et al. (2019) details the creation of the Natural Questions benchmark, but does not specify an annotation time period. A closer inspection of missing DPR titles suggests documents listing information may have been inadvertently excluded[8]. Missing data

[3] See appendix A.1 for more detailed information

[4] Both Sachan et al. (2022a) and Sachan et al. (2022b) utilized the DPR dataset

[5] A.2 presents an example with more information

[6] Section 5 details information on concatenation level

[7] See A.3 for more detailed information on how table metrics were calculated

[8] See Table 6 for a few examples

in the retrieval corpus is a problem, as it's impossible to fetch nonexistent answer passages. Several cited papers publish top-k accuracy on the NQ benchmark utilizing the DPR corpus, but don't specify a missing passage issue and may be biased as a result. To determine if any remediation was taken, we examined the codebase from one such paper. Our evaluation suggests retrieval accuracies were computed by identifying a matching answer span in retrieved passages, which may be problematic as passages may contain the right span, but completely irrelevant to the query. However, due to the excluded answer passages, it's more likely published results are biased downwards, as a subset of queries were most likely recorded as inaccurate failed retrieval cases, suggesting accuracies might be even higher! To address missing data elements in our analysis, we combine unique document information from both DPR and BEIR sources using the article titles. In cases where documents exist in both sources, we elect to use the deduplicated BEIR Wikipedia data.

Source	Total Documents	Missing DPR Documents	Percent Missing
NQ-Train	56,227	6,949	∼12%
NQ-Test	108,593	14,315	∼13%

Table 2: Number of missing document articles from the DPR Wikipedia document corpus.

4.1.2 Final Dataset Details

An augmentation step processes each passage for question generation, which is time intensive on a single GPU[9]. In addition, the retriever requires large amounts of memory during encoding, which restricts the total number of passages we can consider to ∼250,000. To reduce the computational burden we need to reduce the size of the retrieval corpus. First, 20% of the passage/answer pairs were randomly drawn from BEIR NQ-Train. Based on the answer documents in this sample, we randomly select **five** passages from each corresponding document to add similar contextual information as potential sources of "noise". To complete the corpus, we randomly sample from the full Wikipedia document corpus until we reach approximately 250,000 passages. All selected answer passages undergo a question generation step. To address augmentation differences between answer passages and the rest of the corpus sample, we randomly select a subset of unaugmented passages from the newly created corpus for augmentation. Passages not selected were included without adjustment. Table 3 displays summary metrics from the sampling procedure at concatenation level 1[10].

Stage	Source	Size	Sample	Sample %	Augmentations
1	Query Answer Passages (NQ-Train)	132,803	26,634	20%	26,634
2	Answer Document Passages Stage 1 Sample (NQ-Train)	2,613,082	84,040	3.2%	10,734
3	Full Corpus (DPR Data)	19,844,404	149,008	0.75%	15,789
	Total		259,682		53,157

Table 3: Sampling details for the final corpus at concatenation level 1.

4.2 Model and Training Details[11]

Question Generator (QG). To generate potential questions for each document passage, a generative LLM model is used. Instead of tuning from scratch, we leverage an existing resource on the Hugging Face Hub (Montgomerie, 2020b) which allows us to specify the number of generated questions, η. The QG was tuned from the T5-Base (Raffel et al., 2019) on 200,000 examples in SQuAD, CoQA, and MSMARCO datasets. We made several modifications to the existing codebase to improve generation accuracy and reduce inference throughput[12]. To develop a passage specific question set,

[9] See B.1 for throughput details

[10] See Section 5 Question Generation for other concatenation level summaries

[11] Unless explicitly stated, all analysis was conducted on the base Google Colab GPU

[12] See B.1 for throughput information and B.2 for specific enhancements/limitations

the QG attempts to generate η questions, however, passages lacking a formal sentence structure or enough contextual detail result in fewer generated questions[13]. We tested $\eta \in \{25, 50\}$, but realized many passages do not contain enough information for 50 questions. Higher levels of η also reduce the inference throughput and for our experiments we set $\eta = 25$. After generation, each question is ordered by passage relevance score, computed from a separate BERT model (Montgomerie, 2020a) which we did not adjust. The generated questions are delimited and added to the database.

Information Retrieval Model. The information retrieval stage is based on the work by Karpukhin et al. (2020). Given a set of question-passage pairs (q, p), both are fed through pre-trained BERT tokenizers to get their long tensor representations and passed through two independent pre-trained BERT encoders (Facebook, 2020b;a). At this stage, the encoded outputs are float tensors containing dense representations of the initial text inputs. Once encoding of all document passages is complete, the vector representations are indexed for efficient max-inner product search. At inference, a new query is tokenized and fed through the question encoder to get its vector representation q^*. The similarity between the input question representation and a passage is computed as an inner product $\langle q^*, p \rangle$. The top-k indexed passages are retrieved using the FAISS algorithm (Johnson et al., 2017).

It is worth mentioning that encoders and tokenizers were not fine-tuned for our specific modifications. The main reason we did not fine-tune is due to the lack of computational resources in our current setup. Saving relevant tensors in-memory meant we could only work with passages in batches of five, which we further detail in section 5. This limitation translated into each query having a very small number of in-batch negatives, hindering the training procedure. When we attempted to fine-tune, iterating for a single epoch (more than 5,000 batches) took several hours: an unfeasible amount of time to run a full training schedule using limited GPU resources. We decided it was better for our experiments to work with the trained model, developed larger dataset, rather than fine-tuning on much smaller subsample. Also, since the main objective of this work is experimentation and not replication, leveraging the already trained models meant we could run more experiments instead of spending time and computational resources finetuning.

4.3 Results

4.3.1 Information Retrieval Accuracy

We ran several experiments, changing one aspect of the retriever pipeline to compare performance which included: evaluating NQ-Train queries without additional "noise" passages, adding a random sample of 250,000 "noise" passages from the Wikipedia corpus, and adding a random sample of about 250,000 hard "noise" passages from the Wikipedia corpus. Hard noise differs in that passages come from both the Wikipedia corpus and from answer documents in NQ-Train for each sampled query. We assessed retrieval accuracy for these configurations under three concatenation levels and compared performance of the augmented passage set (generated questions appended) against accuracy of the original passages. We also compared two different question generation mechanisms, greedy decoding and beam search. Additionally, we compared the performance of the retriever when the title of the passage is not appended to the passage before encoding. This ablation was tested with and without beam search questions, with and without the hard Wikipedia passages, only for concatenation level 3. Finally, all the configurations were evaluated for top-k passages retrieved with $k \in \{10, 20, 50, 100\}$. The main results are summarised in table 4 with the accuracy averaged across k. A deep-dive for different values of k is presented in section 5.

5 Analysis

Computation Time. While passing passages through the encoder and indexing, some interesting computational tradeoffs were faced. The indexing using FAISS can take batched tensors, and the larger the batch size, the faster the indexes are built. However, the passage encoder quickly exceeds RAM memory, even in large Colab machines, so a balance had to be struck between fast indexing and manageable batch sizes. Our first approach was to save tensors containing the passage embeddings to storage, and later reload them into the index in a different session. This was not an efficient approach because saving and loading all the tensors in-memory quickly saturated RAM.

[13] See B.3 for passage specific examples

Concatenation Level	Questions Appended	Question Types	Wikipedia Passages	"Hard" Passages	Appended Title	Mean Accuracy (%)
1	No	Beam	256,784	Yes	Yes	**77.57**
1	Yes	Beam	256,784	Yes	Yes	70.30
2	No	Beam	259,763	Yes	Yes	**75.74**
2	Yes	Beam	259,763	Yes	Yes	71.36
3	No	Beam	0	-	No	**78.73**
3	Yes	Beam	0	-	No	74.33
3	No	Beam	0	-	Yes	**89.93**
3	Yes	Beam	0	-	Yes	83.89
3	No	Greedy	0	-	Yes	**89.93**
3	Yes	Greedy	0	-	Yes	85.08
3	No	Beam	191,931	Yes	No	59.15
3	Yes	Beam	191,931	Yes	No	**62.52**
3	No	Beam	191,931	Yes	Yes	**73.82**
3	Yes	Beam	191,931	Yes	Yes	72.08
3	No	Beam	250,000	No	Yes	**84.03**
3	Yes	Beam	250,000	No	Yes	80.21
3	No	Greedy	250,000	No	Yes	**83.86**
3	Yes	Greedy	250,000	No	Yes	69.93

Table 4: Summary of retrieval experiments.

At this stage, we were only able to encode and save passages in batches of 5. Later, we switched our approach so that passages were retrieved, tokenized, encoded, and added to the index all in one function call. This meant that we no longer had to store the tensors containing the passage embeddings, only the updated index at each step. With this approach, we managed to feed the passages in batches of 20. Building an index for the passages in NQ-Train took a little under 20 minutes using the regular Colab GPU and a high-RAM session. It is important to mention that we were not able to use the Premium GPUs offered by Colab because the version of the FAISS package available in PIP does not support these GPUs, and we found that installing packages using Conda in Colab sessions to be slow and unstable.

Finally, to embed the Wikipedia passages, we ran some experiments and saw that we could only work with about 250,000 passages before exhausting our computational resources. In order to add these passages to the NQ-Train index, we load the index locally, move it to GPU, tokenize and encode the Wikipedia passages, and add them to the index. As before, we managed to do this indexing step using batches of size 20. In total, adding the $\sim$250,000 passages to each index takes around 2 hours and 45 minutes. Our approach takes advantage of the fact that the same passages belong to different indexes (each for one of our pipeline configurations), and only performs the encoding step one time[14]. To make a fair comparison, we also generate questions for the Wikipedia passages and append them. However, due to time and computational constraints, we are not able to generate questions for the 250,000 passages at each concatenation level.

Question Generation. Several factors govern question generation. Many passages lack a coherent sentence structure, affecting the generation quality as the QG is not able to comprehend poorly structured text. Passage length determines the number of generated questions, as longer passages contain more information and reduce the impact of incoherent text, however, may affect the question topics, as shorter passages may direct the QG to more targeted information about relevant topics. To investigate passage length effects on retrieval accuracy we compare three concatenation levels: 1, 2, and 3. By default, our retrieval corpus divides documents into a series of passages at level 1. For levels 2 and 3, we combine consecutive passages, using the unique document identifier, to collapse two and three passages together within the same document. Passages that could not be combined due to an even or odd number of passages were left as is.

Retriever Performance. Some of the main takeaways from table 4 are that retriever performance is greatly reduced when adding "hard" noise passages, about 10 points for the retriever without questions appended and 8 points for the retriever with questions appended. These "hard" passages come from articles which contain answer passages, so the contents are similar and the retriever

[14] See C for small code examples

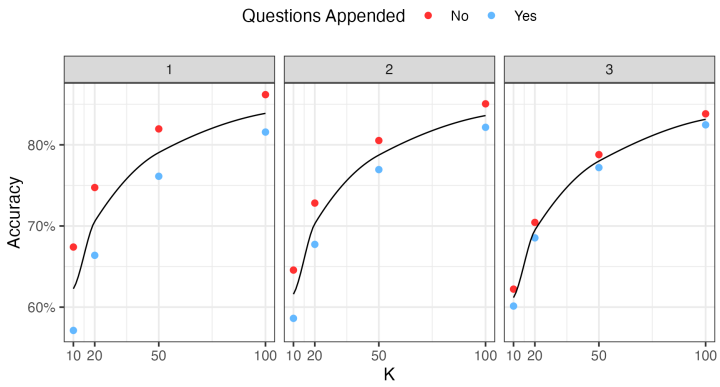

Figure 1: Top-k retriever accuracy for concatenation levels 1, 2, and 3 ($k \in \{10, 20, 50, 100\}$).

fails to capture the correct passage more often. It is also interesting to compare the greedy and beam search question generation. When no noise passages are included, the retriever with greedy decoding performs slightly better than the retriever with beam search. However, when random noise passages are included, there is a big improvement of more than 10 points when switching from a greedy decoding to a beam search decoding strategy. The initial performance difference can be attributed to the duplicating question phenomena under greedy decoding that provides additional keywords for the retriever to match on, which becomes more difficult with additional noise passages. It is also very interesting to note that the performance of the retriever dramatically drops when we remove the article title from the corpus passages. In this case, our method of appending beam search questions performs better than the original retriever. This suggests the retriever is overly reliant on some features which contributes to poor generalizabilty of DPR methods. An interesting idea to explore in future could look to quantify and/or minimize the dependence on any one feature. Overall, we see that the retrievers are very good at performing the task that they were trained to do, but altering the problem slightly may yield big drops in accuracy.

Finally, we compare the performance of retrievers when changing the concatenation level. Figure 1 shows the accuracy for the different values of k when adding "hard" noise passages and appending the titles to the passage. We see that the performance of the retrievers without questions appended decreases as the concatenation level increases, showing once again that the retrievers are better at performing the task they were trained to do. The difference in performance is smaller as k increases. It is also interesting to note that the difference in performance between appending and not appending questions is smaller as the concatenation level increases, and that the performance is better for higher concatenation levels in the case when questions are appended to the passages.

6 Conclusion

In this analysis we introduce an IR method leveraging the concatenation of a generated question set to original document passages. Our results suggest a promising approach, but to improve results we need to fine-tune the passage encoders, which requires significant resource investment. Question generation, while computationally expensive, only needs to be done once. Future work should be directed at scaling up this analysis on a full corpus to reduce variability and spent investigating performance benefits of a tuning based approach. Furthermore, we also performed a variety of experiments consisting of different ablations on the retrievers. Our experiments show that IR tends to perform worse, often significantly, when the task varies from the original implementation, confirming a lack of robustness in DPR strategies.

Author Contributions

Both members contributed equally on this analysis. Andrew curated the dataset and developed the QG pipeline. Jose worked on the retrieval process and evaluation metrics.

REFERENCES

Rajarshi Das, Manzil Zaheer, Dung Thai, Ameya Godbole, Ethan Perez, Jay-Yoon Lee, Lizhen Tan, Lazaros Polymenakos, and Andrew McCallum. Case-based reasoning for natural language queries over knowledge bases, 2021. URL https://arxiv.org/abs/2104.08762.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL https://arxiv.org/abs/1810.04805.

Facebook. facebook/dpr-ctx_encoder-single-nq-base. https://huggingface.co/facebook/dpr-ctx_encoder-single-nq-base, 2020a. Accessed: 2022-12-01.

Facebook. facebook/dpr-question_encoder-single-nq-base. https://huggingface.co/facebook/dpr-question_encoder-single-nq-base, 2020b. Accessed: 2022-12-01.

Gautier Izacard and Edouard Grave. Distilling knowledge from reader to retriever for question answering, 2020. URL https://arxiv.org/abs/2012.04584.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus, 2017. URL https://arxiv.org/abs/1702.08734.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering, 2020. URL https://arxiv.org/abs/2004.04906.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 2019.

Patrick Lewis, Yuxiang Wu, Linqing Liu, Pasquale Minervini, Heinrich Küttler, Aleksandra Piktus, Pontus Stenetorp, and Sebastian Riedel. Paq: 65 million probably-asked questions and what you can do with them, 2021. URL https://arxiv.org/abs/2102.07033.

Adam Montgomerie. iarfmoose/bert-base-cased-qa-evaluator. https://huggingface.co/iarfmoose/bert-base-cased-qa-evaluator, 2020a. Accessed: 2022-12-01.

Adam Montgomerie. iarfmoose/t5-base-question-generator. https://huggingface.co/iarfmoose/t5-base-question-generator, 2020b. Accessed: 2022-12-01.

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, 2019. URL https://arxiv.org/abs/1910.10683.

Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model?, 2020. URL https://arxiv.org/abs/2002.08910.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, apr 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL https://doi.org/10.1561/1500000019.

Devendra Singh Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen-tau Yih, Joelle Pineau, and Luke Zettlemoyer. Improving passage retrieval with zero-shot question generation, 2022a. URL https://arxiv.org/abs/2204.07496.

Devendra Singh Sachan, Mike Lewis, Dani Yogatama, Luke Zettlemoyer, Joelle Pineau, and Manzil Zaheer. Questions are all you need to train a dense passage retriever, 2022b. URL https://arxiv.org/abs/2206.10658.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models, 2021. URL https://arxiv.org/abs/2104.08663.

Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. End-to-end open-domain question answering with. In *Proceedings of the 2019 Conference of the North*. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-4013. URL https://doi.org/10.18653%2Fv1%2Fn19-4013.

A Database Information and SQL Scripts for Computation

A.1 Dataset Information

Both BEIR datasets have identical formats. The source mirror provides jsonl files for corpus (passages) and queries with query answers provided as a tsv file. The data was manually loaded into a GCP cloud storage bucket and uploaded to BigQuery. The resulting schema for NQ-train and NQ-test is shown in figure 2 and was stored in separate databases.

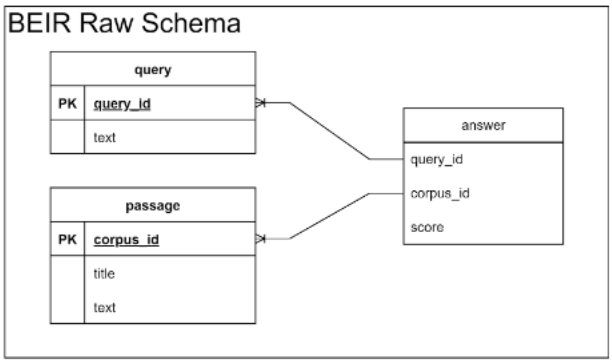

Figure 2: BEIR schema.

A.2 Duplicated BEIR Data

Table 5 shows one example of a duplicated passage in the BEIR NQ training benchmark with original data fields. The raw corpus includes four copies of an identical answer passage with different passage ids, yet only one is marked correct to the NQ query "when's the next resident evil coming out".

Corpus ID	Title	Text
doc532610	Resident Evil (film series)	In May 2017, Constantin chairman Martin Moszkowicz said that a reboot of the film series is in development.[30] James Wan will produce the reboot with a script by Greg Russo.[31]
doc631341	Resident Evil (film series)	In May 2017, Constantin chairman Martin Moszkowicz said that a reboot of the film series is in development.[30] James Wan will produce the reboot with a script by Greg Russo.[31]
doc14251312	Resident Evil (film series)	In May 2017, Constantin chairman Martin Moszkowicz said that a reboot of the film series is in development.[30] James Wan will produce the reboot with a script by Greg Russo.[31]
doc14929455	Resident Evil (film series)	In May 2017, Constantin chairman Martin Moszkowicz said that a reboot of the film series is in development.[30] James Wan will produce the reboot with a script by Greg Russo.[31]

Table 5: A duplicate example in the BEIR benchmark.

To identify these issues, a simple SQL statement selecting distinct title and text fields was used. To compute the summary metrics, we aggregate by title, text and count the number of rows.

```
select title, text, count(*) as cnt
    from beir_nq_train.train_document_lookup
group by title, text;
```

To remediate duplicates at different concatenation levels ω, we assume the minimum passage id at the first position in the `corpus_id_array` to correspond with the correct entry. For $\omega = 2$, the following query removes duplicate entries for identical concatenations.

```
select doc_id, title, text, corpus_id_array
from (
    select A.doc_id, A.title, A.doc_text as text,
        A.corpus_id_array,
        row_number() over(partition by A.doc_id, A.title,
            A.doc_text order by
                cast(substr(A.corpus_id_array[offset(0)], 4)
                    as INT64)) as dupes
    from `beir_nq_train.stg_nq_train_documents_2` A
) A
where dupes = 1;
```

A.3 Missing Answer Passages

Table 6 displays a sample of missing article titles in the DPR dataset that are marked as answering the displayed NQ query. We display only one question from each article for brevity, but some missing articles are referenced across multiple queries.

Missing Article Title	NQ-Train Query
List of Interstate Highways in New York	what are the major highways in new york
List of Green Bay Packers players	who are all the players on the green bay packers
List of tallest buildings	what is the talkest building in the world
Smoking age	whats the legal age to smoke in usa
Malaysian general election, 2018	who won the election in malaysia in 2018

Table 6: A subset of missing article titles with the corresponding NQ-Train query from BEIR.

To identify missing articles in DPR, a more complex SQL query is needed. To reduce parsing errors, titles in both sources are converted to lowercase and strip of spaces. We then compare distinct titles in BEIR to DPR with a join.

```
select A.title from (
  select distinct title from beir_nq_train.train_document_lookup
) A
    left join `nlp_final_project.wikipedia_dump` B
        on REGEXP_REPLACE(lower(A.title), ' ', '') =
            REGEXP_REPLACE(lower(B.title), ' ', '')
where B.title is null;
```

A.4 Final Dataset Details - Additional Concatenation Levels

The encoder is memory constrained and can only handle tables that occupy roughly 300MB of storage, which equates to roughly 250,000 passages at reduced concatenation levels. However, at level 3 we had to reduce the number of randomly sampled Wikipedia passages to fit under the memory limit.

Stage	Source	Size	Sample	Sample %	Augmentations
1	Query Answer Passages (NQ-Train)	132,803	26,634	20%	26,634
2	Answer Document Passages Stage 1 Sample (NQ-Train)	2,034,323	83,795	4.1%	10,083
3	Full Corpus (DPR Data)	10,584,679	151,526	1.4%	16,341
	Total		261,955		53,058

Table 7: Sampling details for concatenation level 2.

Stage	Source	Size	Sample	Sample %	Augmentations
1	Query Answer Passages (NQ-Train)	132,803	26,634	20%	26,634
2	Answer Document Passages Stage 1 Sample (NQ-Train)	1,647,696	82,535	5.0%	11,099
3	Full Corpus (DPR Data)	7,594,345	83,677	1.1%	15,390
	Total		192,846		53,123

Table 8: Sampling details for concatenation level 3.

B Model Details

B.1 Question Generation Throughput

Table 9 shows the record throughput for the QG model on the 26,634 NQ-Train answer passage sample under different conditions. The "QG Optimized" column denotes whether or not the QG utilized speed enhancements detailed in B.2. To facilitate efficient generation, as many as five GPUs were utilized in parallel: three Colab sessions and two local machines. Note the reported "Sample Generation Time" corresponds to running the inference pipeline in only a single session.

Concatenation Level	QG Optimized	Throughput (seconds per passage)	Sample Generation Time (hours)
1	True	5.42	40.09
2	True	8.10	59.94
3	True	10.64	78.70
3	False	13.17	97.41

Table 9: Question generation throughput.

B.2 Question Generator Modifications

Several modifications to the default QG codebase were made. First, to improve the coherence of output questions, the decoding strategy was altered to use beam search (beams = 4) instead of the initial greedy approach. During our experimentation, we also noticed large amounts of redundant questions within the same passage. To address this we introduced a temperature parameter (T=2) to introduce more variability in the sampling distribution, which made outputs more varied. To improve inference speed, we adjusted the default data loader to make use of batching strategy instead of a single record configuration and switched the tokenizer to a "fast" version from Hugging Face.

Lastly, we realized the default generation routine processes passages sentence by sentence, but the original logic was splitting passages by searching only for punctuation symbols. Many passages contain abbreviations like "Mr." or "Mrs.", which affected this parsing strategy. To solve this, we rewrote the splitting scheme using more robust regex and added additional functionality to strip out pesky Wikipedia citations like "[5]" of "[12]" that occupy valuable GPU memory.

We acknowledge the sentence by sentence inference scheme is a potential limitation in the current model architecture that limits the information the QG can reference when performing inference. Currently, the QG generates only one potential question per passage sentence, but this can be adjusted. In future work we plan to alter the number of sentences the QG observes within one forward pass and adjust the number of inferred questions it generates to assess the impact on downstream retrieval accuracy.

B.3 Question Generation Examples

Figure 3 displays an answer passage and the QG outputs under different decoding schemes. The original NQ query was "*in the life of pi who do the animals represent*". Note the amount of question duplication under the "Greedy" decoding scheme and how "Beam Search + Temperature" reduces instances of duplication.

	Contents
Passage	Two hundred and twenty-seven days after the ship's sinking, the lifeboat washes onto a beach in Mexico, after which Richard Parker disappears into the nearby jungle without looking back, leaving Pi heartbroken at the abrupt farewell. The third part of the novel describes a conversation between Pi and two officials from the Japanese Ministry of Transport, who are conducting an inquiry into the shipwreck. They meet him at the hospital in Mexico where he is recovering. Pi tells them his tale, but the officials reject it as unbelievable. Pi then offers them a second story in which he is adrift on a lifeboat not with zoo animals, but with the ship's cook, a Taiwanese sailor with a broken leg, and his own mother. The cook amputates the sailor's leg for use as fishing bait, then kills the sailor himself as well as Pi's mother for food, and soon he is killed by Pi, who dines on him. The investigators note parallels between the two stories. They soon conclude that the hyena symbolizes the cook, the zebra the sailor, the orangutan Pi's mother, and the tiger represents Pi. Pi points out that neither story can be proven and neither explains the cause of the shipwreck, so he asks the officials which story they prefer: the one without animals or the one with animals. They eventually choose the story with the animals. Pi thanks them and says: "And so it goes with God." The investigators then leave and file a report.
Greedy	Where is he recovering [SEP] How long after the ship's sinking [SEP] What does he say to the investigators [SEP] how long after the ship's sinking [SEP] How many parts of the novel are there [SEP] What is the symbol of the hyena [SEP] Who is the captain of the lifeboat [SEP] What is the story's purpose [SEP] What is the story of the shipwreck [SEP] What story is the author's favorite [SEP] Where is Pi [SEP] How does the story end [SEP] What is the story about [SEP] Who kills the sailor [SEP] Who kills the sailor [SEP] Where is he in the hospital [SEP] Where did the lifeboat sink [SEP] What is the story about [SEP] What is the story about [SEP] What is the story about [SEP] Which story is the most likely to be the one without animals [SEP] What is the story about [SEP] What is the story about [SEP] Who is the Japanese Ministry of Transport
Beam + Temperature	How does the investigators conclude that the hyena represents the cook [SEP] How long after the ship's sinking did the lifeboat disappear [SEP] How does he explain the cause of the shipwreck [SEP] How long after the ship's sinking [SEP] Who is responsible for the shipwreck [SEP] Who is the captain of the lifeboat [SEP] Where is he recovering [SEP] Who decides the story with the animals [SEP] How do the investigators note parallels between the two stories [SEP] How does he tell them his story [SEP] Where is he recovering [SEP] Who represents the hyena, the zebra the sailor, [SEP] What does he say to the investigators [SEP] Who kills the sailor [SEP] Who kills the sailor [SEP] What is the story about [SEP] Where did the lifeboat sink [SEP] Which part of the novel describes a conversation between Pi and two officials from the Japanese Ministry of [SEP] How does the story end [SEP] How does the story end [SEP] Where is Pi adrift on a lifeboat [SEP] Which part of the novel describes a conversation between Pi and two officials from the Japanese Ministry of [SEP] Which one does Pi prefer [SEP] Where is he adrift on a lifeboat

Figure 3: A generated question example for one answer passage in NQ-Train.

C Retriever Code Examples

C.1 DataLoader Iterator

```
class MyDataset(Dataset):
    def __init__(self, dataframe, p_tokenizer):
        self.dataframe = dataframe
        self.p_tokenizer = p_tokenizer

        self.p_embed = p_tokenizer(
            self.dataframe['passage_append'].tolist(),
            return_tensors='pt',
            truncation=True,
            max_length=512,
            padding='max_length'
        )

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, index):
        return self.p_embed[index]


def collate_fn(batch):
    batchsize = len(batch)

    ctx_tensor = torch.LongTensor(
        [[sample.ids, sample.attention_mask, sample.type_ids]
        for sample in batch]
        )

    return ctx_tensor

BATCH_SIZE = 20

dataloader_train = torch.utils.data.DataLoader(
    MyDataset(dt_train_clean, ctx_tokenizer),
    batch_size=BATCH_SIZE,
    shuffle=False,
    collate_fn=collate_fn
    )
```

C.2 Creating a FAISS Index

```
res = faiss.StandardGpuResources()
index = faiss.IndexFlatL2(768)
gpu_index = faiss.index_cpu_to_gpu(res, 0, index)

class PassageEncoder(nn.Module):
    def __init__(self, p_encoder, index):
        super().__init__()
        self.p_encoder = p_encoder
        self.index = index

    def forward(self, passage):

        self.index.add(
          self.p_encoder(
           passage[:, 0, :],
           passage[:, 1, :],
           passage[:, 2, :]
          ).pooler_output.contiguous()
        )

pEncoder = PassageEncoder(ctx_model, gpu_index)

for i in tqdm.notebook.tqdm(
dataloader_train, total=len(dataloader_train)
):
    pEncoder(i.to("cuda"))
    torch.cuda.empty_cache()
```