Inter IIT Tech Meet 11.0

Expert Answers in a Flash: Improving Domain-Specific QA
Final Submission

Primary ID: 13
Secondary ID: 26

February 2023

1 Introduction

Domain-Specific Question Answering or DSQA aims to provide precise responses to natural language queries from a repository of external knowledge. One specific use case of such systems is to automate customer support and redirect to human agents only when necessary. Such an intelligent system should be able to offload mundane tasks and make better utilization of the available human resources. Most existing pipelines disentangle this process into two, namely (a) *Information Retrieval* to identify potential candidate contexts from the large knowledge base, and (b) *Extractive Question Answering* to extract one (specific) answer to the given query question. A successful DSQA system must be able to retrieve answers accurately and efficiently for real-time deployment.

Previous works attempt to solve the above two problems individually, and have shown promising results. Text-retrieval methods broadly use dense [1], sparse [2] representations, or a combination of both [3] to represent passages and queries. On one hand, dense representations are more semantically expressive but computationally challenging to construct while sparse features are efficient but are not as feature-rich as their dense counterparts. Previous research primarily focuses on improving retrieval quality, either by introducing several state-of-the-art (SOTA) transformer models (pre-trained) [4, 5, 6], proposing computationally expensive fine-tuning techniques [7] and more recently instruction-tuned large language models in a zero-shot setting [8]. However, these methods are still far from being deployed in real-time use cases. Hybrid approaches attempt to use the best of both worlds but are not mature enough to conduct large-scale retrieval compared to existing search libraries [9]. Methods for extractive question answering leverage pre-trained transformer models and append a classification head to extract answer spans [10]. Although there exists more complex follow-up work that improves performance, the former still continues to be a widely adopted strategy due to its simplicity.

Towards this end, we propose *Retrieve Twice, Rank and Answer*(R2RA), a carefully designed end-to-end pipeline for domain-specific question answering. Deviating from previous research, we primarily focus on efficiency while still maintaining comparable performance against SOTA methods. We conduct several experiments to demonstrate the efficacy of our proposed method. In the following sections, we first motivate our specific design choices in the pipeline, discuss our experimental setup, and finally present our results.

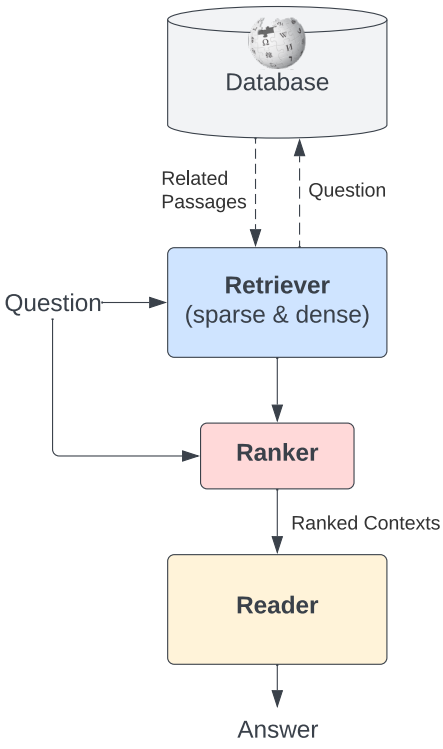

Figure 1: Overview of Retrieve Twice, Rank and Answer (R2RA)

2 Method: Retrieve Twice, Rank and Answer (R2RA)

Overview. Given a set of N paragraphs forming our knowledge base and n questions, our final goal is to extract text spans that answer the given query question. Our method can be divided into three stages: Given a query question, (1) retrieve the top relevant passages (k in number) based on coarse features, (2) optionally re-rank based on fine-grained question-passage similarity, and (3) identify text span from best matching passage retrieved from the previous stages. Unlike few other methods [7], our proposed method carefully builds upon off-the-shelf models without specific fine-tuning that enables it to generalize to a broader range of themes. Optionally, we propose a novel fine-tuning mechanism to derive theme-specific models that preserves knowledge from the pre-trained initialization during transfer learning on the downstream task. Our pipeline is depicted in Fig. 1. Firstly, we use a hybrid combination of dense and sparse embeddings that capture both semantic and lexical relationships to retrieve the closest matching paragraph(s). Our chosen voting scheme demonstrates significant performance improvements over a naive top-k selection from each. This enables us to enjoy the speed benefits without compromising on performance. We also propose a fully automatic strategy to identify potentially answerable questions that can augment the retrieval process. In our second stage, we optionally choose to rerank the retrieved paragraphs from Stage I using a more powerful cross-encoder [11]. Given a fixed set of themes, we finetune these cross-encoders using (just) a given paragraph (unlabelled without corresponding question-paragraph pairs) that further improves re-ranking performance. Finally, the best matching paragraph is passed onto pre-trained question-answering models to predict the start and end indices of the answer span. Throughout our pipeline, we specifically focus on deploying these models on the CPU and therefore limiting our design space from using highly performant but inefficient methods.

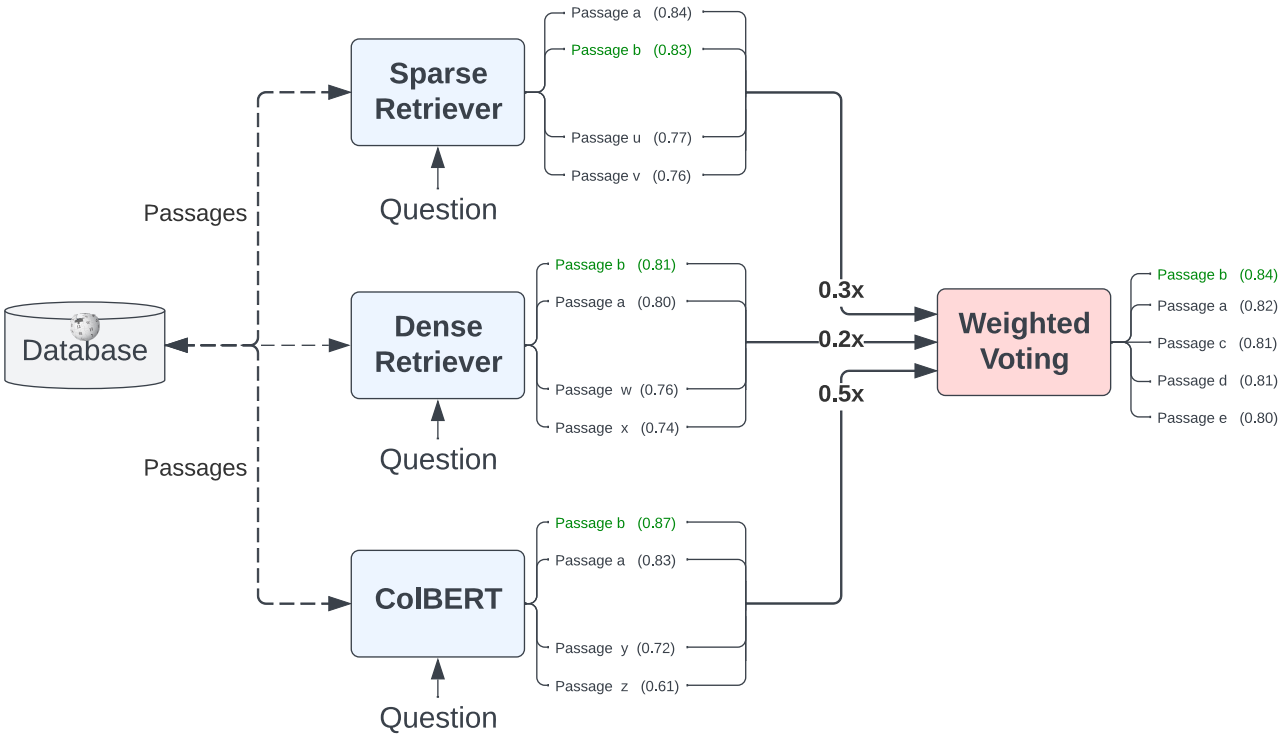

Figure 2: Retriever

2.1 Stage I: Retrieve Twice

In this subsection, we outline our first stage i.e to retrieve the closest matching paragraphs from the knowledge base given a query question. Towards this end, we specifically look at two efficient strategies: *Elastic Search* that leverages lexical similarity to retrieve closest matching pairs, and *Dense Retriever* that focuses on more complex semantic relations between questions, and paragraphs. An ideal retriever must be able to focus on both kinds of information - since smaller, simpler paragraphs could be matched based on exact keyword matches and overall semantic meaning could be matched based on dense embeddings. We further augment our derived representation using embeddings extracted from ColBERT, that models a cheap yet powerful interaction step to encode fine-grained similarity. We ablate on two different aggregation strategies - (a) select the top few each from ColBERT, dense, sparse feature matching and naively concatenate them to retrieve k passages, (b) individually identify confidence scores for each matching type and re-vote based on combined scores to retrieve

closest matching k passages. Fig. 2 illustrates the later strategy. Through several experiments, we identify that the ColBERT embeddings capture semantic information that generalizes across a variety of paragraphs and hence weigh the passages retrieved by the same higher than the dense, sparse ones.

2.1.1 Question-aided Retrieval

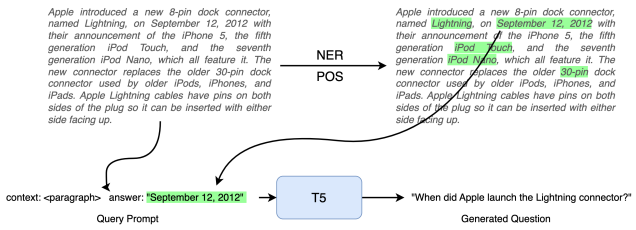

Figure 3: Ranker

High-capacity feature encoders capture a rich representation of the document but might still fail to correlate abstract questions with their related paragraphs - due to the inability to hallucinate complex questions or due to the size of the paragraph tokens. Therefore, we hypothesize that by identifying potentially answerable questions for corresponding paragraphs, one can reuse them for effective retrieval. However, manually generating all possible questions is challenging and we propose a fully-automatic question generation pipeline given any new paragraph.

To generate all possible questions, we derive potential answers by identifying entities, or noun chunks present in the paragraph. We leverage off-the-shelf models for Parts of Speech (POS) tagging and Named Entity Recognition (NER). Using these identified (potentially accurate) answer spans, we appropriately query an LLM model to generate questions. More specifically, we phrase our query prompt as "answer: <answer-span>, context: <paragraph>" and pass it on to a T5 model finetuned on the SQuAD dataset. Using top-p sampling, we identify possible question tokens with a confidence score greater than 0.95. This enables us to derive $\sim$ 130 questions (on average) per paragraph. Fig. 3 illustrates the above process.

2.2 Stage II: Re-Rank

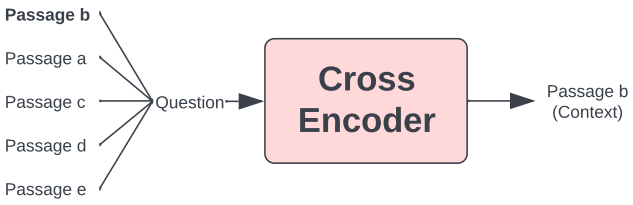

Figure 4: Ranker

In the second stage, we optionally re-rank the retrieved paragraphs using a more complex encoder that captures fine-grained information. Each question is individually concatenated with its corresponding top k retrieved passages and passed onto a pre-trained cross-encoder as input (e.g. MiniLM) to predict a numerical value that estimates its similarity to obtain the best matching paragraph. Fig. 4 describes Stage II figuratively. We do not use the cross-encoder in the first stage as its a computationally expensive procedure to rank multiple paragraphs. Therefore with minimal overhead, we rely on an efficient first stage to retrieve a few potential candidate passages to then retrieve the best. Therefore, our method enjoys the performance benefits of a more complex encoder while still remaining computationally efficient.

2.2.1 SDAFT: Self Distil And FineTune

Given paragraphs from a fixed collection of themes, we further improve the performance of the pre-trained cross-encoder by performing theme-specific finetuning. However, existing fine-tuning methods either choose to finetune all layers, or train a separate linear classifier while freezing the encoder backbone. However, both these methods have certain flaws - the former in the presence of domain drifted data could result in a non-trivial optimum, and the later does not provide us with the best solution. To achieve best performance, one must identify strategies to finetune all the layers such that the intermediate latent distribution remains close to the pretrained model. Towards this end, we propose SDAFT: Self Distill And FineTune - that combines self-distillation and fine-tuning to preserve knowledge from the pretrained model during transfer learning on the downstream task.

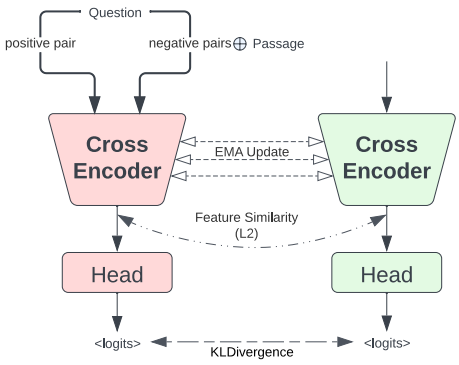

Figure 5: Self Distil and FineTune

We first derive a teacher network using the cloned weights of the pre-trained cross-encoder and distil the final latent feature encoded by the student against the teacher's using mean-squared error loss. After

every training step, we apply a momentum update on the teacher's weights to gradually improve the teacher representation. Given a question and its corresponding retrieved top-5 paragraphs obtained from Stage I, we leverage the incorrectly retrieved pairs as negative samples. This makes sure that the cross-encoder identifies the correct sample amongst closely related question-answer pairs. Fig. 5 describes our overall fine-tuning strategy.

2.2.2 Adaptive ReRanking

To obtain the best performance while still maintaining efficiency, we propose an adaptive strategy to trigger the reranking process. More specifically, if the ensembled confidence scores between the top-1 retrieved and the others is lower than a threshold i.e high similarity between the retrieved paragraphs, we employ the cross-encoder for fine-grained reranking. Otherwise, we skip the re-ranking process to decrease the number of cross-encoder forward calls to improve the overall latency.

2.3 Stage III: Answer

In the final stage, we extract the answer span using the query question and best-retrieved passage. To do so, we concatenate the two and pass them to off-the-shelf pre-trained question-answering models trained to predict the start and end indices of the required span. We experiment with several pre-trained networks and choose the best based on their joint performance and efficiency.

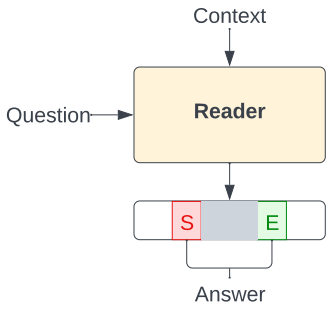

Figure 6: Reader

3 Experimental Setup

Datasets. Deviating from the mid-report (where we derived our own evaluation datasets), we leverage the provided evaluation set containing 944 questions, and 1,179 paragraphs, each broadly classified into one of 14 different themes for fair comparison. During training, we only make use of the paragraphs and not the corresponding question pairs to simulate a more realistic setting (i.e where we only have access to any incoming new paragraph without labelled questions and answer spans). We conduct extensive experiments on the above dataset to verify the performance and efficiency of our method.

Implementation Details. We retrieve the top-5 passages from Stage I, using sparse embeddings based on BM25 [2], dense embeddings encoded using a pre-trained MPNet model [12][1], and a distilled version of ColBERT [13][2] from hugging face. To generate potentially answerable questions, we leverage a T5 [14][3] LLM fine-tuned on the SQuAD dataset. These generated questions are appended onto each paragraph to assist Stage I of retrieval. The top 5 retrieved passages and then passed onto a MiniLMv2 [11][4] to rerank and identify the best matching passage. To obtain theme-specific cross-encoders, we fine-tune each cross-encoder for 10 epochs using our proposed SDAFT strategy. The retrieved paragraph is further passed onto pre-trained tiny-Roberta model [15][5] to identify the relevant answer span. Our midterm report conducts several ablations to identify the best model in each stage that can maximize both performance and efficiency and we directly reuse our findings in this final report.

To further speed up inference, we export these models into their corresponding ONNX formats. ONNX[6], developed by Microsoft, attempts to represent a model, irrespective of its origin framework (e.g pytorch, tensorflow, etc) into one static graph. This enables faster inference times and easy deployment on servers. Further, we attempt to parallelize each stage using multiple threads (at a theme level). Together, along with our efficient implementation of the pipeline helps drastically reduce inferences times by almost half. We cache our entire pipeline using joblib[7] to reduce load times during inference. All latency scores discussed in the following tables use the above strategies to speed up execution (unless otherwise specified).

Metrics. We use widely adopted metrics to evaluate our model's performance and efficiency. In the case of retrieval, we compute the top-1 and top-5 accuracy scores of the retrieved paragraphs against the ground truth, while the final answer prediction is evaluated based on the F1 score between the predicted and ground truth

[1] https://huggingface.co/flax-sentence-embeddings/multi-QA_v1-mpnet-asymmetric-A
[2] https://huggingface.co/sebastian-hofstaetter/colbert-distilbert-margin_mse-T2-msmarco
[3] https://huggingface.co/mrm8488/t5-base-finetuned-question-generation-ap
[4] https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-2-v2
[5] https://huggingface.co/deepset/roberta-base-squad2
[6] https://onnx.ai/
[7] https://joblib.readthedocs.io/en/latest/

Table 1: Performance of R2RA on the Evaluation Dataset

Method	Top-1	Top-5	F1
Sparse BM25	0.701	0.99	-
Dense MPNet	0.730	1.00	-
ColBERT	0.826	1.00	-
w/ Voting Ensemble	0.834	1.00	-
w/ Voting Ensemble (questions appended)	0.854	1.00	-
w/ QA Reader			
deepset/tinyroberta-squad2	-	-	0.794
w/ Re Ranker			
cross-encoder/ms-marco-MiniLM-L2	0.849	-	-
cross-encoder/ms-marco-MiniLM-L2 (fine-tuned)	0.871	-	-
w/ QA Reader			
deepset/tinyroberta-squad2	-	-	0.817

start, end indices. To measure latency, we compute the average time (in ms) over each individual prediction. Latency measurements are carried out on standard Google Colab CPU inference and averaged across multiple runs to account for variances. The final scores are always estimated using the helper function provided by the challenge organizers for consistency.

4 Results

Table.1 discusses the results of R2RA on the provided evaluation dataset.

Discussion on Retriever. The performance of the overall pipeline directly depends on the initial stage, i.e must efficiently rank a large number of paragraphs, and the later stages retrieve the answer span only from the outputs from Stage I. Therefore, we conduct extensive experiments by tuning several hyperparameters. We compare the top-1 and top-5 retrieval accuracies using (a) Sparse embeddings, (b) Dense embeddings (c) ColBERT embeddings, and finally a combination of all three. Our voting strategy outperforms individual retrievers and is even better than several open-source ensembling pipelines. It is interesting to note here that our Stage I retriever achieves a perfect top-5 score (top-5 equals 1.0), implying that the target passage always exists in the retrieved ones. Upon appending each paragraph with potentially answerable questions (generated), we further improve performance top-1 scores by 2% $\uparrow$. This affirms the quality of our automatic question-generation pipeline.

Discussion on ReRanker. Given the top-5 paragraphs from Stage I, we attempt to rerank the predictions in order to further improve top1 accuracy. Surprisingly, we observe that using SOTA cross encoders (without fine-tuning) hurts performance by 1% $\downarrow$. We argue that our Stage I ensembling pipeline is highly performant and a tough baseline to beat using off-the-shelf cross-encoders without finetuning. However, our proposed fine-tuning strategy recovers theme-specific models that significantly improve top-1 scores by 2% $\uparrow$. More specifically, we observe that on heavily domain-drifted themes, for example in the "siachen earthquake" theme, the pre-trained cross-encoder decreases top-1 retrieval scores from 77 to 68 while our fine-tuned cross-encoder improves performance to 80 (3% $\uparrow$). Please note that our cross-encoders are fine-tuned on the generated questions and evaluated on provided question-paragraph pairs.

Discussion on QA Reader. Finally given the best matching paragraph, we attempt to retrieve the answer span that best answers the given query. From our mid-report, we identified that TinyRoberta achieves the best performance without compromising on efficiency and therefore employ the same in our final pipeline. From Table. 1, we observe that the pre-trained QA models perform well across different datasets and improvements to the retriever pipeline directly reflects on improved F1 scores.

Overall, we carefully design individual components that balance both performance and efficiency, and the highlighted methods in Table. 1 constitute our final pipeline. We would like to highlight that we do not require the question-paragraph pairs at any stage of the pipeline and only use the incoming paragraph, and theme label. Our method with the ReRanker outperforms the same without, but at the risk of compromising latency. Therefore, we present both results and argue that the choice really depends on the application. For example, in high-risk question-answering scenarios (e.g providing medical advice), a more accurate model is preferred while otherwise, one could prefer the faster method.

Table 2: Efficiency of R2RA with and without our proposed inference tricks.

Method	Latency (ms) w/o Threading			w/ Threading
	Retriever	QA Reader	Overall	Overall
R2RA (w/o Ranker)	167.014	512.401	679.415	455.316
R2RA (w/ Ranker)	215.461	512.401	727.86	487.588

Discussion on Latency Table. 2 discusses the average time (in ms) to process a query sample, in the order $\mathcal{O}(l^2)$ where l is sequence-length, computed as an average across the evaluation data. These results clearly indicate that our efficient implementation, and other techniques including ONNX conversion, and multi-threading significantly reduce the overall latency to within 50% of the expected inference time (1 second).

5 Discussion on Code Implementation

We provide a heavily abstracted code implementation[8] of our pipeline, with several pre-defined methods for user convenience. In this section, we provide a very high-level usage of our pipeline, given a new paragraph to be added to the knowledge base categorized into a known theme:

- *args = Args(.), args.<argument name>= <new value>*: Defines arguments that govern several aspects of our pipeline, including training, retrieval, question generation, etc.
- *pipe = Pipeline(args), pipe.dump() (or) pipe = Pipeline.load_from_checkpoint(args)*: Defines pipeline with given set of arguments and caches it to reduce load times during inference. Once cached, the pipeline could be directly reloaded using the load_from_checkpoint method.
- *pipe.load_onnx() (or) pipe.load_torch()*: Converts torch models to onnx format for faster inference.
- *generator = GenerateQuestions(args), generator.fit()*: Generates potentially answerable questions for every given paragraph to augment Stage-I retriever and to fine-tune the cross encoder.
- *trainer = ReRankerTrainer(args, df, pipe), trainer.fit(df, pipe)*: finetunes the re-ranker on the given data frame, df to generate theme-wise tuned model checkpoints.
- *pipe(question, theme)*: Performs inference on any incoming question based on the cached knowledge base to obtain the required answer span, along with retrieved paragraph id.

6 Conclusion

In this report, we present our overall pipeline for domain-specific question answering, dubbed Retrieve Twice Rank and Answer or R2RA in short. Our method leverages an efficient ranking strategy to retrieve the top few relevant paragraphs. Further, a reranker is optionally added to re-order the retrieved passages and improve top-1 scores. Finally, the best matching passage is concatenated with the query and passed onto a QA Reader to extract the relevant answer span. R2RA achieves promising results on several datasets containing fewer themes, and generalizes across a broad range of themes and to domain drifted data. Beyond just performance scores, our design choices are driven to improve overall latency on a standard Google Colab CPU.

[8] https://drive.google.com/drive/folders/1IPwB8wNGIVhgzdVeeESuj23SVRzMkMMJ?usp=sharing

References

[1] Vladimir Karpukhin et al. "Dense Passage Retrieval for Open-Domain Question Answering". In: *CoRR* abs/2004.04906 (2020). arXiv: 2004.04906. URL: https://arxiv.org/abs/2004.04906.

[2] Stephen Robertson et al. "Okapi at TREC-3". In: *Overview of the Third Text REtrieval Conference (TREC-3)*. Gaithersburg, MD: NIST, Jan. 1995, pp. 109–126. URL: https://www.microsoft.com/en-us/research/publication/okapi-at-trec-3/.

[3] Luyu Gao, Zhuyun Dai, and Jamie Callan. "COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List". In: *North American Chapter of the Association for Computational Linguistics*. 2021.

[4] Zhenzhong Lan et al. "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *CoRR* abs/1909.11942 (2019). arXiv: 1909.11942. URL: http://arxiv.org/abs/1909.11942.

[5] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[6] Kevin Clark et al. "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators". In: *CoRR* abs/2003.10555 (2020). arXiv: 2003.10555. URL: https://arxiv.org/abs/2003.10555.

[7] Lee Xiong et al. "Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=zeFrfgyZln.

[8] Devendra Singh Sachan et al. "Improving Passage Retrieval with Zero-Shot Question Generation". In: *arXiv preprint arXiv:2204.07496* (2022).

[9] Ruiqi Guo et al. "Accelerating large-scale inference with anisotropic vector quantization". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3887–3896.

[10] Pranav Rajpurkar, Robin Jia, and Percy Liang. "Know what you don't know: Unanswerable questions for SQuAD". In: *arXiv preprint arXiv:1806.03822* (2018).

[11] Wenhui Wang et al. "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5776–5788.

[12] Kaitao Song et al. "MPNet: Masked and Permuted Pre-training for Language Understanding". In: *CoRR* abs/2004.09297 (2020). arXiv: 2004.09297. URL: https://arxiv.org/abs/2004.09297.

[13] Omar Khattab and Matei Zaharia. "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT". In: *CoRR* abs/2004.12832 (2020). arXiv: 2004.12832. URL: https://arxiv.org/abs/2004.12832.

[14] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *CoRR* abs/1910.10683 (2019). arXiv: 1910.10683. URL: http://arxiv.org/abs/1910.10683.

[15] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.