Deep Passage Retrieval in E-Commerce

Vinay Rao Dandin*
vinay.dandin@flipkart.com
Flipkart US R&D Center
Redmond, USA

Ozan Ersoy
ozan.ersoy@flipkart.com
Flipkart US R&D Center
Redmond, USA

Kyung Hyuk Kim*
kyung.kim@flipkart.com
Flipkart US R&D Center
Redmond, USA

ABSTRACT

We have developed a conversational assistant called the Decision Assistant (DA) to help customers make purchase decisions. To answer customer queries successfully, we use a question and answering (QnA) system that retrieves data on product pages and extracts answers. With various data sources available on the product pages, we deal with unique challenges such as different terminologies and data formats for successful answer retrieval. In this paper, we propose two different bi-encoder architectures for retrieving data from each of the two data sources considered – product descriptions and specifications. The proposed architectures beat the baseline approaches while maintaining a high recall and low latency in production. We envision that the proposed approaches can be widely applicable to other e-commerce QnA systems.

KEYWORDS

natural language processing, passage retrieval, neural networks, retriever-reader systems, bi-encoder architecture

ACM Reference Format:
Vinay Rao Dandin, Ozan Ersoy, and Kyung Hyuk Kim. 2023. Deep Passage Retrieval in E-Commerce. In *Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion), April 30-May 4, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3543873.3587624

1 INTRODUCTION

Conversational assistants have become ubiquitous in recent years [1, 11, 12]. In e-commerce, they can help the customer discover products, make a purchase decision, and offer customer support. On our online e-commerce marketplace, we have built a conversational assistant called Decision Assistant (DA) with the goal of helping the customer make the purchase decision similar to an in-person sales assistant in a store.

DA is available on our product pages to answer any product-related query, ranging from questions on product specifications like "What is the battery capacity of this mobile?" to offers and discounts, exchanges, or warranties. DA is available for mobile and fashion verticals in its current iteration. In particular, to answer product-specific questions, two main sources of data are used — product descriptions (called Rich Product Descriptions; RPDs) and product specification key-value stores (KVs). This data is usually provided by product vendors and is considered the most reliable data source at hand. If we are not able to find an answer in RPDs and KVs, we fall back to user-generated content such as reviews.

We approach this question and answering (QnA) task by taking a *retriever* and *reader* approach. The first step is the contextual retriever where we select relevant passages which might contain an answer, given RPDs and KVs. The second step is the reader or the question answering step which identifies answer spans given the subset of passages retrieved by the retriever.

The key challenge around this retrieval task is that the terminology used in the questions and the relevant answers are often significantly different; for example,

Q: "How long can the phone last?"
A: "6000 mAh battery that offers a standby time of up to 57 days."

Q: "What is the resolution of the screen?"
A: "1520x720 Pixels".

Another challenge lies in the nature of different data formats and contents of RPDs and KVs. RPDs mainly consist of descriptive paragraphs of product features while KV stores contain product specifications as key-value pairs. This difference in the data formats and contents give rise to its own set of challenges in the retrieval task. We propose two different model architectures that are uniquely suited for each data source to tackle this challenge.

The rest of the paper is organized as follows: Section 2 contains related work and section 3 discusses the methods we used for the retriever. Section 4 addresses experiments and results. Finally, section 5 contains the conclusion and future work.

2 RELATED WORK

2.1 Passage retrieval

Earlier work on passage retrieval represented documents and queries as sparse vectors, with each dimension corresponding to a term. TF-IDF [8, 18] and BM25 [17] are the popular weighting schemes widely used today. However, these approaches fail to capture the contextual or semantic meaning of the query and documents. To overcome this limitation, latent semantic analysis was proposed by representing documents with low-dimension dense vectors [4].

With the advent of deep learning, mainly two different modeling approaches became popular for retrieval – *bi-encoder* (or dual encoder) and *cross-encoder*. [7] proposed bi-encoder with an embedding-based bag-of-words model where queries and documents were embedded independently, later extended by [15, 19]. Documents are ranked based on their cosine similarity with the

*Both authors contributed equally to this research.

WWW '23 Companion, April 30-May 4, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9419-2/23/04…$15.00
https://doi.org/10.1145/3543873.3587624

query. However, bi-encoder approaches show limitations in capturing fine-grained interactions between the query and documents. To overcome this, [14] and [21] proposed a cross-encoder approach where the query and documents are jointly embedded.

Earlier retrieval systems often took two-step approaches [14, 20]: (1) traditional retrieval based on sparse vector representations [3] and (2) re-ranking based on neural systems [6, 16]. [9] introduced the dense passage retrieval (DPR) which uses low dimensional dense vector embeddings and approximate nearest neighbors to retrieve and rank documents. It's a seminal paper in the field and proposes an efficient technique for training bi-encoders, of which the concept is applied to our batch sampling strategies. For a detailed empirical study on sparse and dense information retrieval systems, please refer to [10].

2.2 Key-Value retrieval

Apart from documents, the retrieval systems can also use knowledge bases (KBs) as their data source. Specifically, [5] proposed key-value retrieval networks for task-oriented dialogue, and [13] proposed key-value memory networks for directly reading both documents and KBs. Our KV task falls in between KB and documents in that both questions and answers can contain either or even both keys and values; for example, "What is the ram size?", "Does this phone come with 2GB ram?", "2GB?". Thus, the various techniques mentioned in this and the previous sections are applicable. Here, the unit of data changes from paragraphs to key-value pairs although the number of tokens in KV pairs is significantly shorter than RPD. In this paper, we use a bi-encoder model for the KV retrieval with keys and values concatenated as the "document".

3 METHODS

3.1 RPD Retrieval

3.1.1 Model Architecture. We implemented a bi-encoder model with two *independent* encoder branches [9, 13] (refer to Figure 1). Here, in each branch the fastText-based sentence embedding vectors were feed-forwarded and concatenated with the bi-LSTM sentence embedding vectors. This feed-forward structure helped increase the model performance compared to the baselines as shown in Table 1.

3.1.2 Model Description. During the model training, relevance probabilities between a query and a set of candidate answers are computed by

$$\boldsymbol{p} = \text{softmax}(q \cdot \boldsymbol{A}^T). \tag{1}$$

Here, q is a query embedding obtained from the query branch and $\boldsymbol{A}$ embedding matrix with $d \times D$ dimensions for answer candidates obtained from the answer branch, with d being the dimension of a batch of answer candidates and D the embedding vector dimension. $\boldsymbol{p}$ is compared with ground truth labels ($\boldsymbol{y}$) and the cross-entropy loss is computed based on $\boldsymbol{p}$ and $\boldsymbol{y}$. Please note that the embedding from each branch is the concatenation of the fastText and bi-LSTM sentence embedding vectors and thus its dimension D is the sum of both embedding vector dimensions. The underlying assumption in the softmax is that there is only one relevant paragraph within each training batch of size d. In the next section, we describe in detail how we ensured this assumption in the batch sampling.

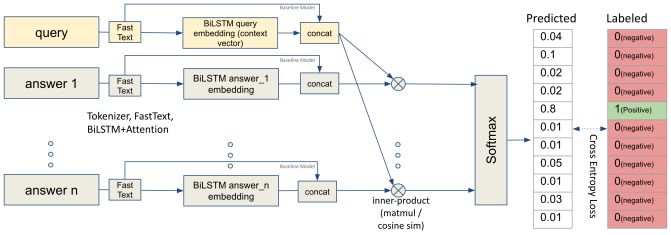

Figure 1: Model architecture for the RPD retrieval: a bi-encoder with two independent encoder branches was used. For both the branches, each text input was split into tokens, converted into fastText word embedding vectors, and used as input for bi-LSTM. The output of the bi-LSTM was concatenated with the fastText sentence embedding – an embedding vector averaged over the fastText word embedding vectors of all the sentence words.

During inference, when a user types in a query in DA, a set of RPDs from a product page where the user is on are treated as answer candidates (Figure 2). The user query serves as input to the query branch and the RPDs as input to the answer branch returning query and answer embedding vectors respectively. The inner product between the query and each answer embedding vector is computed and the paragraphs corresponding to top-k inner products are selected as output. Here, the main task is to retrieve the most relevant paragraphs with high recall.

3.1.3 Batch Sampling During Training. During the model training, each batch of size d (= number of user queries) is prepared by randomly selecting d queries from the train data set which have unique intents, i.e., no duplicate-intent queries. In this way, we made sure that there is only one positive pair existing in each batch when paired with all the answers available within each batch. For example, the first query in Fig. 3 is related to the finger_print_unlock intent while all the other queries are to different intents and the same logic holds for all the other queries. Thus, this way of negative sampling strategies will guarantee one positive sample in our Softmax computation [6, 9]. The loss function for each batch was computed by using the following relevance probabilities:

$$\boldsymbol{p} = \text{softmax}(\boldsymbol{q} \cdot \boldsymbol{A}^T) \tag{2}$$

where $\boldsymbol{p}$ is a $d \times d$ matrix, $\boldsymbol{q}$ is a $d \times D$ matrix, and $\boldsymbol{A}$ is a $d \times D$ matrix. The softmax function is applied to each row of $\boldsymbol{q} \cdot \boldsymbol{A}^T$. In this way, we efficiently computed the loss function with full vectorization. For the intent annotation, we used our in-house intent prediction model.

3.2 Key-Value Retrieval

3.2.1 Model Architecture. We took a supervised learning approach and have done multiple experiments with different model architectures/weighting schemes and the final one showing the best performance was a bi-encoder with a bi-LSTM query branch and fastText simple-average answer branch (Fig. 4 and Table 2).

3.2.2 Model Description. During the model training, relevance probabilities between a query and a set of candidate answers (here,

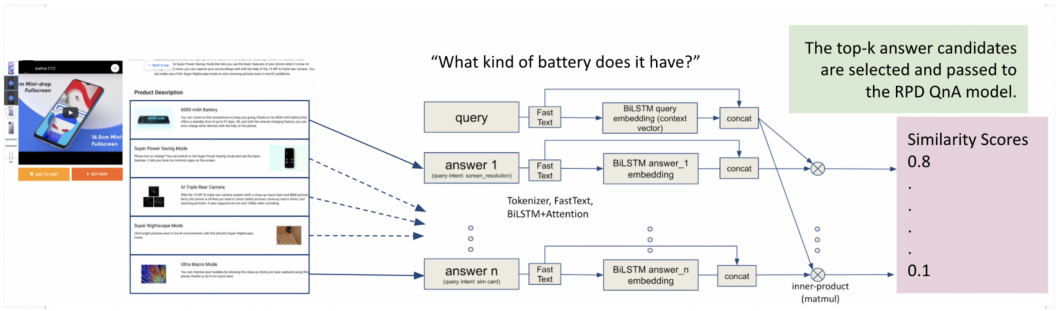

Figure 2: Inference pipeline for the RPD retrieval

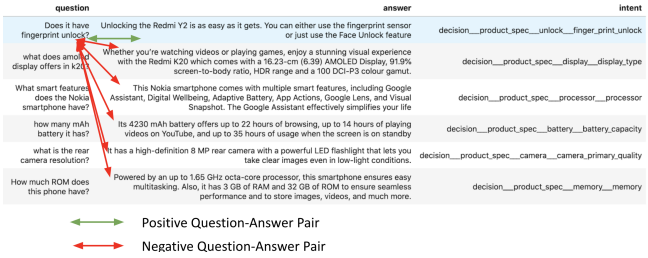

Figure 3: RPD train batch example: For each user query, we ensure that there is one positive answer (paragraph) and $d-1$ negative answers by pulling unique intent queries, i.e., duplicate intents are not allowed within each batch.

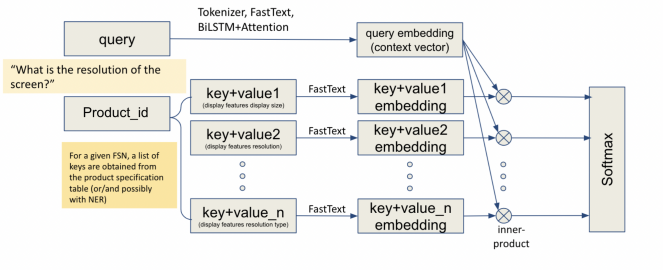

Figure 4: Model architecture for the KV retrieval: A user query and a set of key-value pairs are processed by the query and answer branches, respectively. Both the user query and each of the key-value pairs (key and value are concatenated) are tokenized and converted into fastText word embedding vectors. The word embedding vector for the query is processed by the bi-LSTM encoder, while that for the key-value pair is simply averaged over the tokens.

key-value pairs) were computed by Eq. 1. Please note that the crucial difference from the RPD model is that only the query branch is trained (supervised learning) but the answer branch is not. Since the input to the answer branch is a set of key-value pairs comprising a few words, bi-LSTM is unnecessary. Thus the answer embedding is computed by averaging the fastText embedding vectors of each key and value word along with other features such as the header of the KV table. The cross-entropy loss function is computed in exactly the same way as the RPD case. During back-propagation, the model weights in the query branch are adjusted so that the query embedding vectors become similar/dissimilar to the answer embedding vectors for the positive/negative pairs (refer to Fig. 4).

During inference, a set of key-value pairs from KV on a user-landing product page are treated as answer candidates, which are taken as input to the answer branch, returning answer embedding vectors. The inner product between the query and each answer embedding vector is computed and the key-value pairs corresponding to top-k inner products are selected as output.

3.2.3 Batch Sampling During Training. We took a different batch sampling approach for KV; for a given pair of (product_id, query), we assumed that there is only one relevant key-value pair in the product specification table of the product_id, and assumed all the other key-value pairs as negative. Considering there are typically 100 keys per product_id, this sampling strategy provided a large number of negative samples with many variations of key-value pairs. Thus, the product specification intents were not utilized for the sampling strategy for this KV retrieval. For each batch, 64 (product_id, query) pairs were considered.

4 EXPERIMENTS AND RESULTS

4.1 Data sets used

The data used for training all models (for both RPD and KV retrieval tasks) was refactored from the labeled set prepared for the downstream *reader* steps: span-based QnA. Since we know where an answer to a specific question is we could also easily extract the ground truth paragraphs where the answer lies in.

For both the RPD and KV retrieval tasks, the data sets were then split into training and validation sets while keeping the products unique in each set. This was done to prevent any feature leak and to ascertain the model was not trained on the same context (RPDs and KVs) that also exists in the validation data set. We reserved 4 product_id's as a test data set, which includes 347 (77 + 92 + 83 + 95) labeled question-answer pairs for RPD retrieval and 789 labeled question-answer pairs for KV retrieval. While the size of the test data set seems low, we ensured enough diversity in the labeled question-answer pairs.

For the RPD retrieval task, this exercise resulted in about 2000 samples in training and about 400 records in the validation set.

Requiring more data to train the model, the training set was augmented using paraphrasing and synthetic question generation. After the augmentation, the final training set size became around 13,500 records.

For the key-value retrieval task, our labeled data set consists of 3,277 records for training and 782 records for validation. We did not deploy augmentation techniques for this task; we had more data to start with and the task is somewhat more relaxed (k is higher in this case) and sample efficient.

4.2 Baseline Setup

4.2.1 RPD Retrieval. As a baseline model, we implemented a sparse vector approach based on the BM25 algorithm [17] using ElasticSearch. We indexed RPD paragraphs for all product_id's in ElasticSearch and used its BM25 implementation to retrieve relevant RPDs given a user query. As a baseline for the dense vector approach, an unsupervised bi-encoder based on fastText embeddings [2] was used. In each encoder, we averaged fastText word embedding vectors weighted by TF-IDF [8, 18] scores for both a user query and candidate RPDs and rank the cosine similarity scores. The fastText model is trained with user-generated content (such as user reviews and FAQs), product catalog, and human-agent chat data. Further, we had a third model where we used a pre-trained bi-encoder RoBERTa sentence transformer[16] to encode user queries and RPDs. These models showed limited performance when user queries are significantly different from the corresponding answers in terminology.

A high-level requirement of the task was that it needed to be fast enough to not overload the end-to-end pipeline. Thus, we explored simpler models using the LSTM architecture and did not explore the transformer models further.

4.2.2 KV Retrieval. For this retrieval task, we used an unsupervised bi-encoder model as well similar to the one mentioned above albeit with a small difference. Specifically, the *query* embedding is constructed by averaging fastText word embedding vectors weighted by TF-IDF scores [8, 18], while the answer embedding by taking a simple average over the fastText word embedding vectors (bi-encoder (fastText) in Table 2). This unsupervised model showed limited performance since the embedding vector spaces for user queries and their answers can be significantly different from each other.

4.3 Evaluation Metrics

For both the retrieval tasks, performance was measured by Recall @ top k which considers a prediction as a success if the ground truth RPD passage or KV pair is retrieved in the top k. For RPD retrieval, we decided $k = 3$ for maximal recall because $k = 4$ and higher can make the retrieved texts too long to deal with the downstream BERT-based QnA model, which extracts an *answer span* from the retrieved paragraphs. On a similar note, we decided on $k = 20$ for KV retrieval.

4.4 Results

4.4.1 RPD Retrieval. Table 1 shows the bi-encoder models with the fastText-embedding feed-forward loop provide better recall than the one without feed-forward.

%	Roberta-STS	Elastic-search (BM25)	bi-encoder (fastText)	bi-encoder (bi-LSTM)	bi-encoder (bi-LSTM + feedforward)
Top@1	-	-	76.37	89.34	87.32
Top@2	-	-	88.76	94.81	96.25
Top@3	86.1	88.2	93.08	97.98	**98.27**
Top@4	-	-	94.81	98.85	100

Table 1: RPD passage retrieval: Top k recall for the baseline models and two different bi-encoder models.

%	bi-encoder (fastText)	bi-encoder (value-only)	bi-encoder (key-value-header concat)
Top@1	60.17	65.90	70.60
Top@2	77.35	80.14	83.02
Top@3	85.71	86.69	89.23
Top@20	95.06	97.71	**98.10**

Table 2: Key-value retrieval: Top k recall for a variation of bi-encoder models. In bi-encoder (value-only and key-value-header concat), a user query is processed by bi-LSTM and the corresponding answer branch input with TF-IDF-weighted average over fastText word embedding vectors.

We experimented with different relevance probabilities:

$$\boldsymbol{p} = \text{softmax}[(q\boldsymbol{\Phi}) \cdot (\boldsymbol{A}\boldsymbol{\Phi})^T]. \quad (3)$$

where $\boldsymbol{\Phi}$ is a D × D matrix, transforming each query/answer candidate embedding vector leading to more flexibility for the model to learn each other's embedding spaces. But, this additional flexibility did not help improve the recall values (not reported in this manuscript).

The measured latency (95% percentile) in CPU was ~ 50 msec/query and in GPU (NVIDIA V100) ~ 20 msec/query.

4.4.2 KV Retrieval. For the key-value retrieval task, we experimented with different answer encoding schemes: (1) Key-value-header concatenation (2) key-value concatenation without header (3) key embedding vector concatenated with value embedding vector (4) value-only embedding vector. In these variations, we kept the query encoder as bi-LSTM. Among these experiments, the best performing one was (1). The metrics for (2) and (3) were very similar to (1) and we decided on (1) as it is expected to have the additional information from the header texts. The model performance metrics are reported in Table 2. (1) and (4) are reported in the last and middle columns respectively.

KV retrieval had a similar requirement on latency as the RPD retrieval task. The measured latency (95% percentile) in CPU was ~ 50 msec/query and in GPU (NVIDIA V100) ~ 20 msec/query.

5 CONCLUSION AND FUTURE WORK

In this paper, we explored supervised passage retrieval models with the bi-encoder architecture which have been deployed for mobile phones and fashion products. The retrieval models have a high recall & low latency, and are being used in production at scale. Here, we experimented with different bi-encoder architectures and show that simple architectures can achieve high performance for downstream

tasks. These bi-encoder models are also sample-efficient and can be trained very quickly making it suitable for e-commerce use cases. As we expand Decision Assistant towards millions of products other than mobile phones and fashion products, the retrieval models that we discussed here will play an even bigger role in the retrieval tasks and we look forward to developing them further by minimizing model supervision and using new data sources like user-generated content.

REFERENCES

[1] Janarthanan Balakrishnan and Yogesh K Dwivedi. 2021. Conversational commerce: Entering the next stage of AI-powered digital assistants. *Annals of Operations Research* (2021), 1–35.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the association for computational linguistics* 5 (2017), 135–146.

[3] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051* (2017).

[4] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science* 41, 6 (1990), 391–407.

[5] Mihail Eric and Christopher D Manning. 2017. Key-value retrieval networks for task-oriented dialogue. *arXiv preprint arXiv:1705.05414* (2017).

[6] Daniel Gillick, Alessandro Presta, and Gaurav Singh Tomar. 2018. End-to-end retrieval in continuous space. *arXiv preprint arXiv:1811.08008* (2018).

[7] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 2333–2338.

[8] Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* (1972).

[9] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906* (2020).

[10] Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. 2021. Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics* 9 (2021), 329–345.

[11] Anirban Majumder, Abhay Pande, Kondalarao Vonteru, Abhishek Gangwar, Subhadeep Maji, Pankaj Bhatia, and Pawan Goyal. 2018. Automated assistance in e-commerce: An approach based on category-sensitive retrieval. In *European Conference on Information Retrieval*. Springer, 604–610.

[12] Helen McBreen. 2002. Embodied conversational agents in e-commerce applications. In *Socially Intelligent Agents*. Springer, 267–274.

[13] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. arXiv:1606.03126 [cs.CL]

[14] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).

[15] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. 2016. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24, 4 (2016), 694–707.

[16] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[17] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at TREC-3. *Nist Special Publication Sp* 109 (1995), 109.

[18] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management* 24, 5 (1988), 513–523.

[19] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd international conference on world wide web*. 373–374.

[20] Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. 2019. Multi-passage bert: A globally normalized bert model for open-domain question answering. *arXiv preprint arXiv:1908.08167* (2019).

[21] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-end open-domain question answering with bertserini. *arXiv preprint arXiv:1902.01718* (2019).