Open-Domain Question Answering with Pre-Constructed Question Spaces

Jinfeng Xiao[†], **Lidan Wang**[*], **Franck Dernoncourt**[*], **Trung Bui**[*], **Tong Sun**[*], **Jiawei Han**[†]
[†]University of Illinois at Urbana-Champaign
{jxiao13,hanj}@illinois.edu
[*]Adobe Research
{lidwang,franck.dernoncourt,bui,tsun}@adobe.com

Abstract

Open-domain question answering aims at locating the answers to user-generated questions in massive collections of documents. Retriever-readers and knowledge graph approaches are two big families of solutions to this task. A retriever-reader first applies information retrieval techniques to locate a few passages that are likely to be relevant, and then feeds the retrieved text to a neural network reader to extract the answer. Alternatively, knowledge graphs can be constructed and queried to answer users' questions. We propose an algorithm with a novel reader-retriever design that differs from both families. Our reader-retriever first uses an offline reader to read the corpus and generate collections of all answerable questions associated with their answers, and then uses an online retriever to respond to user queries by searching the pre-constructed question spaces for answers that are most likely to be asked in the given way. We further combine one retriever-reader and two reader-retrievers into a hybrid model called R^6 for the best performance. Experiments with large-scale public datasets show that R^6 achieves state-of-the-art accuracy.

1 Introduction

Open-domain question answering, abbreviated as *OpenQA* in this paper, aims at enabling computers to answer user-submitted natural language questions based on a large collection of documents (a.k.a. a corpus). There are two big families of state-of-the-art OpenQA algorithms. One family, namely retriever-readers (Fig. 1, left branch), first retrieves from the corpus some documents or paragraphs that are likely to be relevant to the question, and then uses neural networks to read the retrieved passages and locate the answer. Another line of work, namely question answering using knowledge bases (abbreviated as *QA using KB* in this paper; Fig. 1, middle branch), first constructs a knowledge base (KB) from the corpus, then queries the KB with the given question. Either family of algorithms has some pros and cons: all retriever-readers face a trade-off between efficiency and accuracy; QA using KB methods are good at answering simple factoid questions within the KB schema but weak at complex or out-of-schema questions.

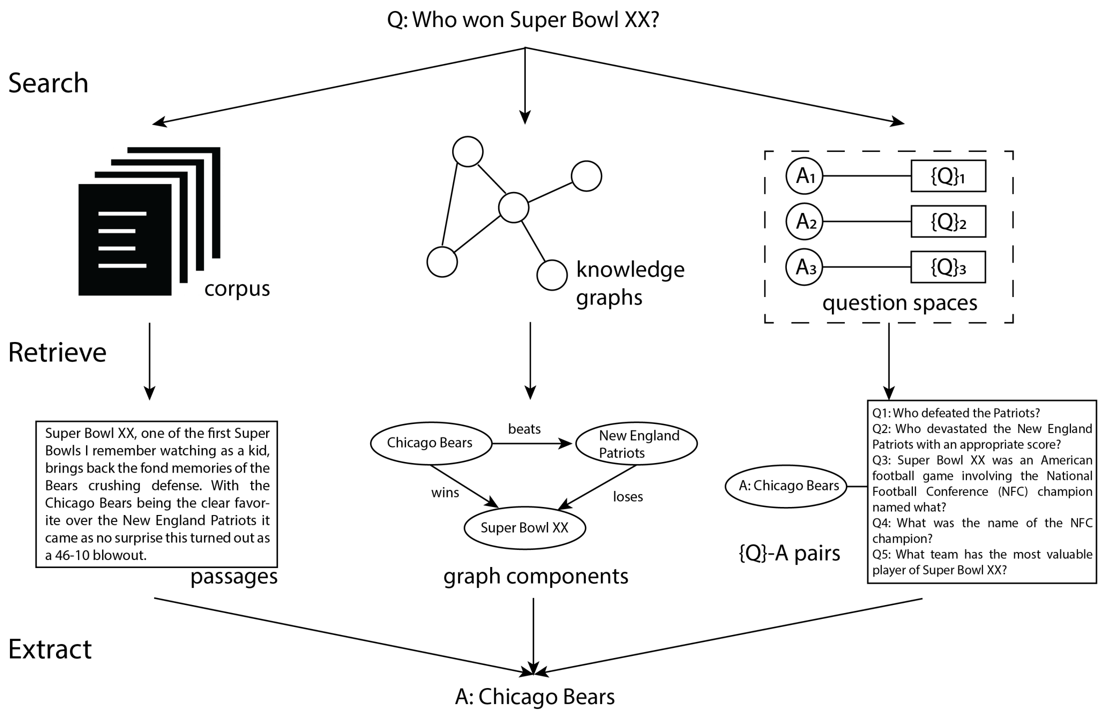
We propose a novel reader-retriever design for OpenQA (Fig. 1, right branch). First, we use deep neural networks to read the corpus offline, detect named entities, generate questions, and aggregate the results into two collections of questions that are answerable with the corpus. We use *question spaces* to term the two collections. When users submit queries online, a retriever compares user queries with the pre-constructed question spaces to retrieve the answers that are most likely to be asked in the given way. We combine two reader-retrievers (one for each question space) and one retriever-reader into a hybrid model called R^6 to predict the most likely answer based on the consistency among the three sub-models. Experiments with large-scale public datasets show that the pre-constructed question spaces boost the performance for OpenQA, and R^6 performs better than state-of-the-art methods by a large margin. The source code of R^6 is publicly available at https://github.com/JinfengXiao/R6.

2 Related Work

2.1 Retriever-Readers

Retriever-readers solve OpenQA by converting it to easier single-passage QA tasks. Examples of popular algorithms in this family include DrQA (Chen et al., 2017), which has a TF-IDF retriever followed by a recurrent neural network reader, and BERTserini (Yang et al., 2019), which consists of a BM25 retriever and a BERT reader.

All retriever-readers face a trade-off between efficiency and accuracy. When the retriever module is



Super Bowl XX, one of the first Super Bowls I remember watching as a kid, brings back the fond memories of the Bears crushing defense. With the Chicago Bears being the clear favorite over the New England Patriots it came as no surprise this turned out as a 46-10 blowout.

Q1: Who defeated the Patriots?
Q2: Who devastated the New England Patriots with an appropriate score?
Q3: Super Bowl XX was an American football game involving the National Football Conference (NFC) champion named what?
Q4: What was the name of the NFC champion?
Q5: What team has the most valuable player of Super Bowl XX?

Figure 1: Retriever-readers (left), QA using KB (middle), and reader-retrievers (right).

computationally efficient, the retrieved results are not very reliable, and the performance of the subsequent reader is also constrained (Htut et al., 2018). On the other hand, there exist systems such as R^3 (Wang et al., 2018) and DS-QA (Lin et al., 2018) that have sophisticated retrievers jointly trained with the readers, but they are computationally expensive and thus not scalable to large corpora (Das et al., 2019).

2.2 QA Using KB

There are solutions that solve OpenQA with knowledge bases (KB). QA using KB applications include Google Knowledge Graph and Bing Satori (Uyar and Aliyu, 2015). Such approaches involve an offline knowledge graph construction module and an online graph query module. The graph construction module scans the corpus to build a knowledge base that contains one or more knowledge graphs. Each graph usually involves some types of entities, attributes and relations. Once a knowledge base is constructed, OpenQA tasks can then be converted to graph search tasks, which can be done in various ways including template decomposition (Zheng et al., 2018) or graph embedding (Huang et al., 2019).

There are a lot of challenges remaining for QA using KB. Examples include how to convert complex natural language questions into structured KB queries, how to alleviate error propagation from the KB construction step to the graph query step, and how to handle questions whose answers do not fall within the KB schema. Due to those complexities, the community is observing a recent trend that retriever-readers are dominating the leaderboards of public QA datasets but KB-based methods are not. Therefore, we choose to focus on the comparison with retriever-readers when experimentally evaluating our proposed algorithm.

3 Approach

3.1 Question Spaces

Definition 1. A *question space* is a bipartite graph with two disjoint and independent node sets A and Q representing the answers and associated questions. We herein define two types of question spaces: QA Spaces and {Q}A (read as Q-set-A) Spaces. In a *QA Space*, each element $a_{i,j}$ of A represents the jth mention in the corpus of the ith distinct named entity, and each element $q_{i,j}$ of Q is a question generated from the context of $a_{i,j}$ with a_i as its answer. For every i and j, $a_{i,j}$ and $q_{i,j}$ form a *QA pair* and are connected in the graph. In a *{Q}A Space*, each element a_i of A represents the ith distinct named entity, and each element q_i of Q is a collection of the $q_{i,j}$'s for all j in the QA Space. For every i, a_i and q_i form a *{Q}A pair* and are connected in the graph. In short, a QA space contains pairs of answer mentions and generated questions, while a {Q}A space contains pairs of distinct answer entities and collections of all generated questions with that answer.

For example, given the five questions in the right branch of Fig. 1 whose answer is “Chicago Bears”, the QA Space will have five QA pairs: $\{a_{1,1}$ = “Chicago Bears”, $q_{1,1}$ = “Who defeated the Patriots?”}, ..., $\{a_{1,5}$ = “Chicago Bears”, $q_{1,5}$ = “What team has the most valuable player of Super Bowl XX?”}, and the {Q}A space will have one {Q}A pair: $\{a_1$ = “Chicago Bears”, q_1 = {“Who defeated the Patriots?”, ..., “What team has the most valuable player of Super Bowl XX?”}}.

3.2 Algorithm

A detailed illustration of our algorithm is given in Figure 2. The components above the grey dashed line are offline. They construct the QA Space and the{Q}A Space as defined in Definition 1. The modules below the grey dashed line are all executed online.

3.2.1 NER, Question-Generating Reader and Question Aggregator

Given a corpus, a named entity recognition (NER) tool called TAGME (Ferragina and Scaiella, 2010, 2012) is applied to detect named entities from the corpus and link the entities to Wikipedia titles. Those entities form the set of candidate answers A in Definition 1. Then a question-generating (QG) reader is applied to the set of candidate answers to generate a question for each answer based on the local context. This reader features an encoder-decoder model structure with a question-answering reward and a question fluency reward tuned with policy gradient optimization (Yuan et al., 2017; Hosking and Riedel, 2019). Then we use a question aggregator to build the {Q}A Space by putting together all the questions with the same answer entity.

3.2.2 Passage Retriever and QA Reader

Given a query, the passage retriever uses the dot product of the query embedding and passage embedding vectors generated by Google Universal Sentence Encoder (Google USE) (Cer et al., 2018) to retrieve from the corpus a passage that is semantically most similar to the query. We then use BERT (Devlin et al., 2019), fine-tuned on SQuAD, to read the retrieved passage, predict the answer, and record the predicted answer as *Answer 1*. The pipeline in Figure 2 that goes from Input Corpus to Passage and then Answer 1 is a valid retriever-reader workflow, and we denote this workflow as **Retriever-Reader-BERT-Large** or **Retriever-Reader-BERT-Base**, depending on which BERT model is used.

3.2.3 Individual Question Retriever

Given a query, the individual question retriever uses Google USE to retrieve from the QA space k questions that are semantically most similar to the query. We record the ordered list of answers associated with the top k retrieved questions as *{Answer 2}*. A majority vote (where ties are resolved by average orders) over {Answer 2} can produce a single answer denoted as *Voted Answer 2*. Then the pipeline in Figure 2 that goes from Input Corpus to Candidate Answers, QA Space, {Answer 2}, and finally Voted Answer 2 (not shown in the figure) is a valid reader-retriever workflow. We denote this workflow as **Reader-Retriever-QA-Space**.

3.2.4 Aggregated Question Retriever

Given a query, the aggregated question retriever uses the BM25 score (Robertson and Zaragoza, 2009) to retrieve from the {Q}A space the answer whose associated set of questions is most similar to the given query. We query the {Q}A Space by treating each q_i as a single document which contains $q_{i,j}$ for all j as sentences. In practice, we observe that BM25 works better for long documents and Google USE works better for short passages. That is why we use BM25 as the aggregated question retriever but use Google USE for the passage retriever and the individual question retriever. We record the answer a_i associated to the top-ranked question set q_i as *Answer 3*. The pipeline in Figure 2 that goes from Input Corpus to Candidate Answers, QA Space, {Q}A Space and finally Answer 3 is a valid reader-retriever workflow. We denote this workflow as **Reader-Retriever-{Q}A-Space**.

3.2.5 Answer Aggregator

Now that we have Answer 1, {Answer 2}, and Answer 3, the last step is to aggregate them into one single answer to return to the user. Our answer aggregation works as follows: if Answer 1 appears in the set {Answer 2}, then accept Answer 1 and return it; otherwise reject Answer 1 and return Answer 3. In other words, the answer aggregator checks the consistency between the retriever-reader results and the reader-retriever ones, trust the retriever-reader more if they agree to some extent,

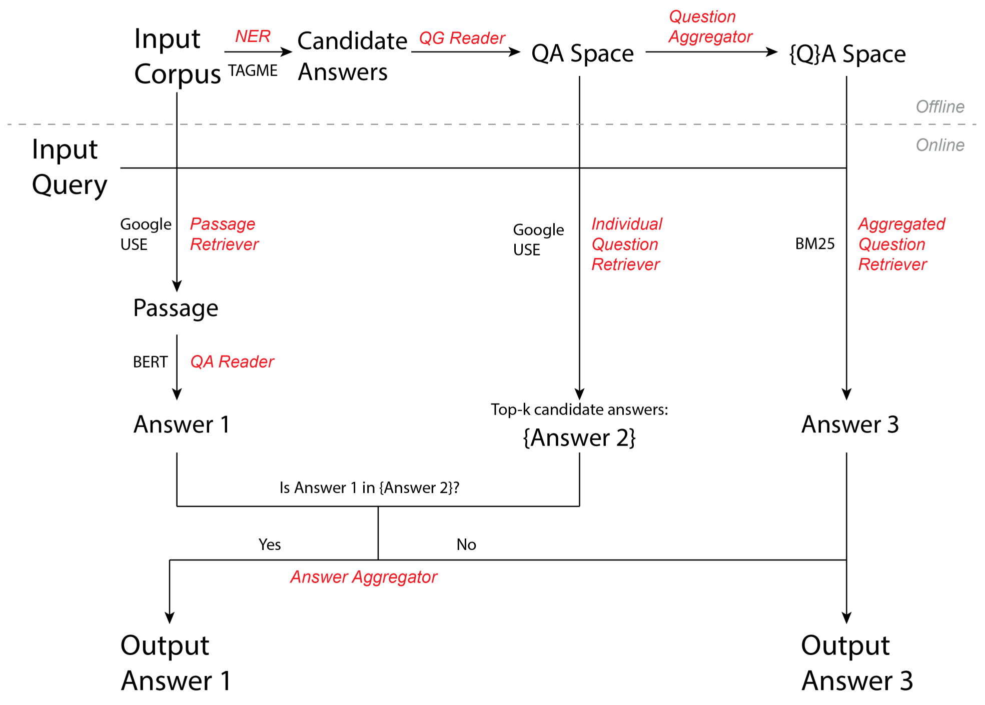


Figure 2: Detailed structure of the proposed method.

and trust the reader-retriever more if the results do not agree at all. We denote the complete workflow depicted in Figure 2 as **R^6**.

4 Experiments

We evaluate the OpenQA performance of our proposed method R^6 and baseline methods using two public QA datasets, SQuAD (Rajpurkar et al., 2016) and TriviaQA (Joshi et al., 2017). We adopt a rather challenging setting that all trainable components of the models are trained on SQuAD, while the final models are tested on TriviaQA. Furthermore, we use TriviaQA in an open-domain setting by removing all annotated associations between questions and documents and enforcing the systems to answer every question with the entire corpus. We write **TriviaQA-Open** to distinguish such an open-domain setting from those officially adopted by TriviaQA.

One may wonder why we choose to use different datasets for training and testing. Because our goal of the experiments is to compare the effectiveness of our proposed methods to others, as long as all the methods are evaluated fairly under the same setting, we can achieve the goal. Such experimental settings are also used by the authors of DrQA (Chen et al., 2017). In addition, using SQuAD for training enables us to utilize pre-trained models and author-suggested hyper-parameters to the greatest extent, so that we can make sure we correctly reproduce others' work and do not put their models into disadvantages when comparing them with ours. More experimental details are available in Section 4.2. Although not critical to this study, using different datasets for training and testing has one additional benefit that it shows the ability of the systems to adapt to new corpora.

4.1 Models

We evaluate six different OpenQA methods with the exact match accuracy in the predicted answers on TriviaQA-Open. Five of them are introduced in Section 3, and the other is DrQA as introduced in Section 2. Here we summarize the basic structure of all six methods in Table 1.

4.2 Reproducibility Notes

This section aims at providing as many details as possible that are needed to reproduce our results. All experiments are run on an Ubuntu 16.04 machine with eight GeForce GTX 1080 GPUs (CUDA version 10.1) and 24 CPUs. The entity score threshold for TAGME is set at 0.2 by tuning that value and manually inspecting the NER quality for 20 documents sampled from TriviaQA. The k value for the individual question retriever that generates {Answer 2} is set to 10. For TriviaQA, we treat each paragraph with at least 50 characters as a passage,

Table 1: Model structures. Arrows show orders of modules.

Model	Description
R^6	Two reader-retrievers + Retriever-Reader-BERT-Base
DrQA	TF-IDF retriever $\rightarrow$ RNN reader
Retriever-Reader-BERT-Large	Google USE retriever $\rightarrow$ BERT-large reader
Retriever-Reader-BERT-Base	Google USE retriever $\rightarrow$ BERT-base reader
Reader-Retriever-QA-Space	QG reader $\rightarrow$ Google USE retriever
Reader-Retriever-{Q}A-Space	QG reader $\rightarrow$ Question aggregator $\rightarrow$ BM25 retriever

Table 2: Test accuracy on TriviaQA-Open. Columns are explained in Section 4.3.

Method	Accuracy	Proposed vs SOTA	Complete vs Components
R^6	**0.30**	●	●
DrQA	0.18	●	
Retriever-Reader-BERT-Large	0.16	●	●
Retriever-Reader-BERT-Base	0.15	●	●
Reader-Retriever-QA-Space	0.07		●
Reader-Retriever-{Q}A-Space	0.21		●

and drop paragraphs shorter than that. BERT is downloaded from the pytorch-transformers GitHub repository[1] and fine-tuned on SQuAD following the documentation. The question-generating reader is obtained from the question-generation GitHub repository[2] and trained on SQuAD with default settings. DrQA codes are downloaded from its GitHub repository[3], the model trained by the authors on SQuAD is obtained as instructed, and the hyperparameter n-docs is set to 1 at prediction time for fair comparisons with R^6. The Google USE retrievers are implemented by re-ranking the top one thousand BM25-retrieved passages with dot products between Google USE embedding vectors obtained with TensorFlow[4].

4.3 Overall Test Accuracy

Table 2 reports the overall test accuracy on TriviaQA-Open of our proposed method R^6, three state-of-the-art methods (DrQA, Retriever-Reader-BERT-Large, and Retriever-Reader-BERT-Base), and the two novel workflows we introduce (Reader-Retriever-QA-Space and Reader-Retriever-{Q}A-Space). The column "Proposed vs SOTA" indicates which rows to look at for comparing our method with state-of-the-art OpenQA methods, while the column "Complete vs Components" indicates which rows to look at for analyzing the contribution of each individual component to the complete model R^6.

Our proposed method R^6 outperforms both DrQA and BERT by a margin six times larger than that between DrQA and BERT. If the 2% difference between DrQA and BERT represents the consequence of differences in the detailed design of the retriever and reader modules in a retriever-reader model (e.g. TF-IDF vs semantic embedding, RNN vs BERT), then the 12% margin between R^6 and DrQA should be largely credited to the essential differences in the overall model structures.

When individual components of R^6 are inspected, our novel reader-retriever component on the {Q}A Space also outperforms DrQA and BERT, with a smaller margin though. Our reader-retriever component on the QA Space is not working well by itself, but as an integral part of the answer aggregation mechanism, it helps push up the performance of our complete model R^6.

4.4 Test Accuracy for Various Answer Types

We further examine how the discussed algorithms work for different answer types. Following the same practice as in the TriviaQA paper (Joshi et al., 2017), we sample 200 question-answer pairs from TriviaQA-Open and manually analyze their properties. We find that about 36% of those questions

[1] https://github.com/huggingface/transformers
[2] https://github.com/bloomsburyai/question-generation
[3] https://github.com/facebookresearch/DrQA
[4] https://tfhub.dev/google/universal-sentence-encoder/2

Table 3: Test accuracy on TriviaQA-Open-200 for various answer types.

Method	Overall	Person/Org (36%)	Location (26%)	Others (38%)
R^6	**0.34**	**0.56**	**0.46**	0.05
DrQA	0.22	0.33	0.23	**0.11**
Retriever-Reader-BERT-Base	0.20	0.39	0.23	0
Reader-Retriever-QA-Space	0.10	0.22	0.08	0
Reader-Retriever-{Q}A-Space	0.22	0.28	0.38	0.05

have person names or organization names as answers, 26% ask for locations, and 38% are expecting other types of answers including entities with other types, numbers, and other free texts. This sample distribution is roughly consistent with what TriviaQA authors have reported (32%, 23%, and 45% respectively) with their random sample. We then use this sampled dataset **TriviaQA-Open-200** to evaluate the test accuracy of the methods for different answer types. We drop Retriever-Reader-BERT-Large for this analysis because its overall accuracy is very close to Retriever-Reader-BERT-Base (Table 2) but it consumes much more computational resources.

The results of this experiment are shown in Table 3. Among the three types, questions that ask for Person/Organization names or locations look significantly easier to answer for all algorithms than those asking for other miscellaneous things, and our proposed method R^6 takes the lead. Among the other models, it looks like BERT is good at questions about Person/Organization names and our newly proposed reader-retriever algorithm on the {Q}A Space is good at answering questions for locations. On the other hand, when the expected answer is neither a person/organization nor a location, DrQA still has some chance of getting the right answer, while all other methods including ours almost always fail. This is probably due to the fact that our methods rely on NER (Figure 2) but DrQA does not. It is possible that better NER methods that are good at handling miscellaneous entity types and numbers could further boost the performance of R^6, and how to better answer those miscellaneous questions is left for future work.

4.5 Notes on Question Space Quality

A manual inspection into the constructed question spaces revealed three aspects worth discussion. 1) Many questions look reasonable, and those generated questions shown in Figure 1 are actually real examples taken from our {Q}A Space that are associated with the answer "Chicago Bears". 2) There are also many questions that to some extent deviate from being a "correct" question to ask for a given answer. One frequently observed mistake is the use of a wrong question word. 3) Some highly context-dependent questions like "who did Bob talk to" are generated. Although they are reasonable and answerable given the context, they do not really make sense when being asked in an open-domain setting. Since R^6 relies on the generated questions, its performance is hopeful to get further enhanced if the quality of the question spaces can be improved. How to generate better question spaces for OpenQA remains an interesting future direction.

5 Conclusion

We propose R^6, a novel algorithm that constructs question spaces from corpora and uses them to improve OpenQA. R^6 consists of two novel reader-retriever modules and one classic retriever-reader. Experiments on public datasets show that R^6 outperforms state-of-the-art retriever-readers by a large margin. Our method has the potential to get further improved if solutions can be proposed in future work to better handle questions about less typical answer types or generate questions with higher quality.

References

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–

1879, Vancouver, Canada. Association for Computational Linguistics.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, and Andrew McCallum. 2019. Multi-step retriever-reader interaction for scalable open-domain question answering. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Paolo Ferragina and Ugo Scaiella. 2010. TAGME: on-the-fly annotation of short text fragments (by wikipedia entities). In *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*, pages 1625–1628. ACM.

Paolo Ferragina and Ugo Scaiella. 2012. Fast and accurate annotation of short texts with wikipedia pages. *IEEE Softw.*, 29(1):70–75.

Tom Hosking and Sebastian Riedel. 2019. Evaluating rewards for question generation models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2278–2283, Minneapolis, Minnesota. Association for Computational Linguistics.

Phu Mon Htut, Samuel Bowman, and Kyunghyun Cho. 2018. Training a ranking function for open-domain question answering. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 120–127, New Orleans, Louisiana, USA. Association for Computational Linguistics.

Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*, pages 105–113. ACM.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Yankai Lin, Haozhe Ji, Zhiyuan Liu, and Maosong Sun. 2018. Denoising distantly supervised open-domain question answering. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 1736–1745. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Stephen E. Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

Ahmet Uyar and Farouk Musa Aliyu. 2015. Evaluating search features of google knowledge graph and bing satori: Entity types, list searches and query interfaces. *Online Inf. Rev.*, 39(2):197–213.

Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerry Tesauro, Bowen Zhou, and Jing Jiang. 2018. R^3: Reinforced ranker-reader for open-domain question answering. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 5981–5988. AAAI Press.

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. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 72–77, Minneapolis, Minnesota. Association for Computational Linguistics.

Xingdi Yuan, Tong Wang, Caglar Gulcehre, Alessandro Sordoni, Philip Bachman, Saizheng Zhang, Sandeep Subramanian, and Adam Trischler. 2017. Machine comprehension by text-to-text neural question generation. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 15–25, Vancouver, Canada. Association for Computational Linguistics.

Weiguo Zheng, Jeffrey Xu Yu, Lei Zou, and Hong Cheng. 2018. Question answering over knowledge graphs: Question understanding via template decomposition. *Proc. VLDB Endow.*, 11(11):1373–1386.