Domain-Specific Question Answering at Scale for Conversational Systems

Sina J. Semnani[†, ‡], **Madhulima Pandey**[‡], and **Manish Pandey**[*]

[†]Stanford University
[‡]Sparrow Labs
[*]Carnegie Mellon Univerity

sinaj@stanford.edu, madhulima@sparrowlabs.ai, mpandey@andrew.cmu.edu

Abstract

Answering questions from documents is a foundational building block of conversational systems. End-to-end open-domain question answering (QA) systems answer natural questions from a large collection of documents. However, the need to search for answers in a broad range of specialized domains ranging from IT infrastructure to health sciences remains a challenging problem. Many of these domains lack extensive labeled training datasets. Besides, the QA system must provide real-time responses during the dialog. In this paper, we present Mindstone, a domain-specific question answering system. This system consists of a multi-stage pipeline that employs a traditional BM25-based information retriever, RM3-based neural relevance feedback, neural ranker, and a transformer machine reading comprehension stage. Mindstone also incorporates techniques that, starting with a small labeled domain-specific dataset, use data augmentation to improve the accuracy of question answering and allow Mindstone to generalize to new domains. Mindstone greatly outperforms state-of-the-art methods for end-to-end open-domain question answering from Wikipedia articles. Our domain adaption techniques significantly improve pipeline accuracy for new domains.

1 Introduction

Answering questions from documents and text passages is a foundational building block of conversational systems. During a conversation, users ask natural questions that can be answered as a continguos text span from passage based on the question and the conversation history. In this paper, we introduce Mindstone, a domain-adaptive question answering (QA) system, which answers user's questions from large collection of documents. We present results for QA over Wikipedia as well as domain-specific questions in the area of enterprise data warehouse [1]. While this paper focuses on single-turn QA, the techniques described can also be applied to multi-turn conversational systems.

One of the first significant open-domain QA system was DrQA [7] that combined term-based information retrieval techniques with a multi-layer RNN-based reader to identify answers in Wikipedia articles. In more recent work, BERTserini [24] integrates the Anserini retriever [23] with BERT [8], to obtain large improvements over prior results for answering questions from Wikipedia. While BERT and other pre-trained transformer models have enabled machine comprehension systems to reach state of the art performance on paragraph-level datasets (SQuAD EM/F1=89.7/92.2 [2]), the overall QA retriever-reader pipeline score is about half this number.

We have improved the end-to-end preformance of QA systems by using a BERT-based neural ranker with RM3-based neural relevance feedback. This has enabled Mindstone achieve a new state-of-the-art QA pipeline peformance on Wikipedia by a significant margin. In this paper, we also show how a

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

very small domain-specific labeled dataset combined with automated data augmentation and dataset creation and a neural ranker allows QA systems to generalize to questions in new domains.

In the rest of the paper, we discuss background and related work in Section 2. Section 3 describes the specialized data warehousing related QA dataset we use for our experiments. Section 4 describes the key components of the Mindstone QA pipeline. The experimental setup and results are desribed in Sections 5 and 6, respectively.

2 Background and Related Work

The machine reading task of answering questions from a piece of text has made great progress in recent years. There are two primary reasons for this. The first one is the creation of datasets like QACNN/DailyMail [9], WikiQA [25] and SQuAD [16] and the QAngaroo multi-hop comprehension database [20], and the CoQA [17] conversational question answering dataset. The second is the considerable progress in deep learning architectures like attention-based and memory augmented neural networks [4, 21] , Transformers [18] and pre-trained models [8]. In this paper, we focus on the problem of Open-Domain Question Answering (QA) which involves searching though a large collection of documents for the span of text in one or more documents that best answers the user's question. While our work focuses on single-turn question answering, the retrieval and ranking techniques described our work can also be applied to multi-turn conversational systems.

Until recently, open-domain question answering has been mostly addressed through the task of answering from structured knowledge bases such as WebQuestions [5] and Simple Questions [6]. However, KB limitations such as incomplete or missing information and fixed schemas, and the recent advances in machine reading have generated new interest in question answering from a large corpora of unstructured documents. New datasets such as MS MARCO [13] where questions sampled from real anonymized user queries are paired with real web documents from the Bing search engine. The recently released Natural Questions [11] dataset includes real anonymized Google search queries paired with human-annotated answers from Wikipedia pages. Natural Questions includes two types of answers: long (an HTML tag that includes the answer) and short (a span of text). The datasets that we use in this paper (SQuAD and our data warehousing dataset) are similar to the short answers.

The DrQA system [7] answers questions from the entire Wikipedia. Its pipeline combines a document retriever and a bidirectional RNN paragraph reader. While the paper employs a document-level retriever based on bigram hashing and TF-IDF matching, our experience has been that a BM25-based paragraph-level retriever with article titles prepended to paragraphs performs better.

Yang et al. [23] have published BERTserini, which uses a paragraph-level Anserini-based retriever and a fine-tuned BERT reader for answering questions. In a follow-on work [24], Yang et al. discuss a data augmentation technique using distant supervision that collects more answerable as well as unanswerable paragraph-question pairs to train the reader. While this work established the previous best results on open-domain QA, it does not discuss any techniques for domain adaptation. Furthermore, as our experiments show, adding a neural ranker to the pipeline results in better overall performance at a higher speed.

3 Data

3.1 Corpora

For open-domain question answering, Wikipedia and Snowflake text corpora serve as our knowledge sources. We use the same 2016-12-21 English Wikipedia dump as [7], which has more than 5 million articles and 37 million paragraphs (Figure 3).

As a domain-specific corpus for IT infrastructure and data warehouse domain, we have chosen the publicly available documents for Snowflake Computing, a cloud data warehousing company, at `snowflake.com`. We crawled the publicly available webpages of Snowflake with a web crawler that processed static and dynamic pages, as well as PDF documents. Images, videos and non-text content were ignored. These were processed into documents with titles, sections and paragraphs and converted into a searchable database and index. This corpus contains 2,105 documents, with a total of 14,254 paragraphs and an average of 74 words per paragraph.

Context: snowflake clients initiate every connection to a snowflake service endpoint with a handshake that establishes a secure connection before actually transferring data. as part of the handshake, a client authenticates the tls ssl certificate for the service endpoint. the revocation status of the certificate is checked by sending a client certificate request to one of the ocsp online certificate status protocol servers for the ca certificate authority.

Question: how do snowflake clients initiate connection to a snowflake service endpoint

Answer: with a handshake that establishes a secure connection

Context: download and import the latest snowflake gpg public key from the snowflake web interface or the public keyserver ... for example, to download the snowsql installer where bootstrap_version is 1.1 and version is 1.1.82: bootstrap_version version the macos operating system can verify the installer signature automatically, so gpg signature verification is not needed. windows s3 url the s3 url pattern for windows is as follows: __code_fragment__ for example, to download the snowsql installer where bootstrap_version is 1.1 and version is 1.1.82

Question: is gpg signature verification needed for macos?

Answer: verification is not needed

Figure 1: Examples from Snowflake dataset

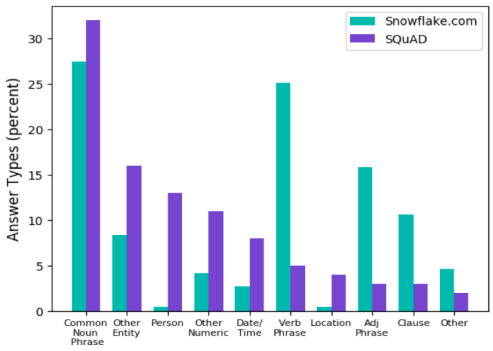

Figure 2: Answer types in Snowflake dataset

3.2 Labeled datasets

We use SQuAD version 1.1 to train the document reader. We use BERT ranker trained on MS MARCO from [14], and we finetune it further with a SQuAD-based augmented dataset.

Using data warehouring glossary terminology, and search logs, (Context, Question, Answer) tuples in a similar fashion to SQuAD version 1.1 were created. In this dataset, each question has exactly one answer, as opposed to SQuAD where answers come from three different people. Two examples are shown in Figure 1. A total of 644 question-answer pairs were created from 174 articles, and 443 paragraphs. The train and the development sets have 488, and 156 questions respectively.

Figure 2 shows answer types of questions from Snowflake compared to SQuAD. It is instructive to see that for an IT platform, the most common answer types are noun-phrase, verb phrase and adjective phrase, which are common responses to system status queries, or how-to documents.

4 Approach

4.1 Mindstone pipeline architecture

In this section, we describe the Mindstone pipeline. Following [23], we first convert the corpus articles into paragraphs. We will use paragraphs and documents interchangeably to refer to the resulting paragraphs. We also prepend article titles to each paragraph to provide some context from the full article. For each query[1], all documents travel through all stages of the pipeline. Figure 3 depicts these stages and their relative order.

Assume the corpus has N^{corpus} paragraphs. In theory, retriever, ranker and reader each assign a score to every text span of every paragraph based on the user query. We use $S^{retriever}$, S^{ranker} and S^{reader} to denote these scores. In practice, all text spans in the same paragraph have the same retriever and ranker score. Also, we only calculate S^{ranker} for top $N^{retriever}$ documents (when sorted by retriever score) to lighten the job of the slower ranker. The final score of a span of text is a weighted average of its three scores, where the weights are tuned to maximize the exact match metric on a small subset of the training set. We normalize all scores to have a value in $(-\infty, 1]$ before taking the average.

4.1.1 Retriever

We use a TF-IDF-based retriever to retrieve $N^{retriever}$ documents. More specifically, we use Anserini [22] based on Lucene version 8.0, and Okapi BM25. We use Anserini to index the documents as well. Our index only considers unigrams that are not in a predefined set of stop words.[2] The main advantage of this component is its speed, and it needs to have a high recall@$N^{retriever}$ since the performance of the full pipeline is upper bounded by it.

[1] We use query and question interchangeably.

[2] Adding bigrams for a paragraph-level corpus slightly hurts the pipeline's performance and speed. DrQA uses unigrams *and* bigrams for indexing, which improves the performance on an article-level corpus.

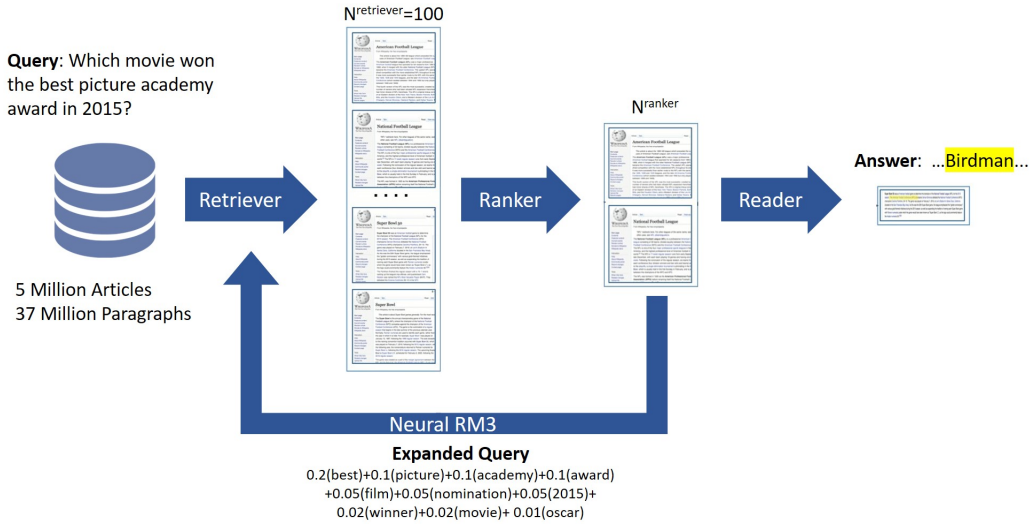

Figure 3: Mindstone pipeline architecture. Numbers are for the Wikipedia corpus.

4.1.2 Ranker

We use a BERT-base ($110M$ parameters) model for ranking the retrieved documents. The model is trained as a binary classifier on MS MARCO using the same method as Nogueira and Cho [14], and then fine-tuned on SQuAD. For more details on different fine-tuning approaches, see section 5. S^{ranker} is the output of the classifier before softmax layer. Our ranker takes the first 448 tokens of the paragraph and the whole query as input and outputs whether the paragraph contains an answer to the query.

4.1.3 Neural RM3

Our initial experiments showed that using RM3 [12] in retriever slightly hurts its recall. Instead, we use a method similar to RM3, but using ranker document scores instead of retriever scores. Let $d_1, ..., d_{N^{retriever}}$ denote the sorted output of retriever and $S_1^{ranker}, \geq ... \geq S_{N^{retriever}}^{ranker}$ be the scores that ranker has assigned to them. Let $v(d)$ be the vector of TF-IDF scores of the top T most common terms in document d where T is a hyperparameter. Then $q' = \alpha q + (1-\alpha)\Sigma_{S_i^{ranker}>0} v(d_i)$, where q is the original query vector and $\alpha \in [0, 1]$ is a hyperparameter, will be the expanded query. We use this new query to retrieve more documents from the corpus and feed them through ranker. This is intended to increase retriever's recall. In the open-domain SQuAD setting, neural RM3 increases recall@100 by 6 points, but improves the exact match of the full pipeline only by 0.5 points at the expense of a slowdown. Therefore, in section 6 we report our results without neural RM3. We believe this method can be especially useful for multi-hop question answering, and leave it to future work.

4.1.4 Reader

The reader finds the exact location of the answer in the ranked documents. We assume the answer is a contiguous span of text and use the approach in [8] , i.e. we use a linear layer for two token-level classification tasks to determine the start and the end position of the answer span. Paragraph and query are truncated or padded to be exactly 384 tokens in total. Since we are using version 1.1 of SQuAD, the reader always returns an answer. We experiment with both base and large BERT models.

4.2 Query generation

To make better use of very small domain-specific QA datasets such as Snowflake, Mindstone has a query generation component that given an unlabeled corpus, automatically creates more (Paragraph, Query, Answer) tuples. We train a BERT-base model to detect keywords (potential answers to the queries that will be generate) in paragraphs, then use the query generator model in [19] to generate queries from the full Snowflake corpus.

Table 1: Examples of query generator's input (paragraph) and outputs (query and answer span).

Paragraph	Extracted keyword	Generated query
The identifier for the managed account is not the same as the account name, which is required to access the account ...	not the same as the account name	what is the identifier for the managed account of the managed account ?
3.15 Release Notes March 1, 2019. This article provides a brief overview of ...	March 1, 2019	when was the release of 3.15 ?
Our tests showed Snowflake's automatic concurrency scaling improved overall concurrent query performance by up to 84% ...	84%	what was the rate of concurrent query performance ?

We use Spacy [3] to split the paragraphs of the SQuAD dataset into sentences, then create a new dataset consisting of (p, s, k) tuples where p is a paragraph from SQuAD, s is a sentence from that paragraph and k is the answer span in that sentence. We train a BERT-base model to detect the start and end position of k from s given p. The training procedure and hyper-parameters are identical to training a reader on SQuAD. We then use this model to predict keywords from each sentence in the Snowflake corpus. To improve the quality of keywords and since not all sentences contain a keyword, we remove keywords that the the keyword extractor is uncertain about, i.e. their scores are lower than some threshold. We then use the trained LSTM-based model from [19] to generate questions from the corpus and the extracted keywords. Note that we fine-tune the keyword extractor on the small Snowflake training set, but not the query generator since we speculate its small non-pre-trained model is not data-efficient enough to benefit from such a small dataset.

In the end, we obtain 50,596 new queries. See Table 1 for a few examples. Most of the generated queries are not grammatically or semantically correct, but if used in conjunction with the training set, can improve the reader's and pipeline's performance.

5 Experiments

We have trained all models on the training set of the mentioned datasets. Since SQuAD's test set is not publicly available, we use a small subset of its training set for development, and report the results on its development set.

5.1 Fine-tuning the ranker

We train the ranker on MS MARCO, and then fine-tune it on SQuAD. We experimented with three different approaches:

1. (Fine-tuning) Convert the SQuAD dataset into a binary classification task where the goal is to determine whether or not a paragraph contains the answer to the question. This is done as follows: for every (Paragraph, Question) pair in SQuAD, choose another paragraph from the same Wikipedia article that does not contain the answer string.
2. (Data augmentation 1) Use the retriever to retrieve more relevant documents from the corpus. More specifically, retrieve n documents for each question in the dataset, and add them to the new dataset. Their labels are determined according to whether or not they include the answer string.
3. (Data augmentation 2) Similar to the second approach, use the retriever to obtain m paragraphs, then rank them and use the top n results.

We use $m = 100$ and $n = 5$ for our experiments.

[3] `https://spacy.io`

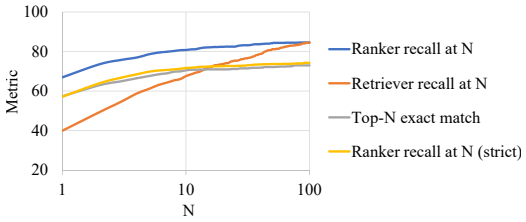

Figure 4: Retriever and ranker recall. The x-axis is log-scale.

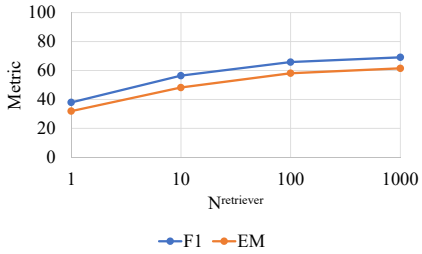

Figure 5: Performance improves logarithmically with num. retrieved documents

5.2 Fine-tuning the reader

For SQuAD, we use the same approach as Devlin et al. [8]. We train for 2 epochs with learning rate $3 \cdot 10^{-5}$ and batch size of 18.

For Snowflake, we use query generation to make up for the small size of the dataset. We start from the reader trained on SQuAD, then fine-tune it on the generated dataset first. A final fine-tuning is done on the Snowflake training set. This multi-step approach is more effective than training on a mixture of generated and original queries. To prevent over-fitting, we freeze every third layer of BERT during training, and change which layers are frozen after each epoch. We use learning rate 10^{-6} and train for 50 epochs with batch size 18.

6 Results

Unless stated otherwise, we use the same metrics (exact match, F1 and recall) as defined in [7]. $N^{retriever}$ is set to 100 for SQuAD to be comparable to prior work, and 50 for Snowflake due to its smaller corpus size.

6.1 Retriever and ranker performance

In Figure 4, retriever and ranker recalls show how much ranker improves the retrieval process.

Mindstone outputs a sorted list of answers, so we can calculate accumulative exact match and F1. Top-N exact match in Figure 4 shows the probability of finding at least one exact-match answer in the top N answers.

We also analyze the performance gap between retriever and the full pipeline. Prior work and other sections of this paper all use the exact match metric in the same way: if a paragraph includes the gold answer string, it is a hit in terms of recall. However, this approach can overestimate recall for queries that have a common phrase such as a number or a person's name as their answer. We introduce a more strict approach to calculating recall. We consider a retrieved paragraph to be a hit only if it is the same as the paragraph that crowdworkers used to write the query in the process of building SQuAD. Since the version of Wikipedia dump we are using is different from that of SQuAD's (and other processing differences), a simple equality check would not work. We use Python's difflib module to assign a similarity score in the range $[0, 1]$ between retrieved paragraphs and SQuAD paragraphs. We consider paragraphs that have a similarity score above 0.1 and contain the answer string to be a hit. Using this definition, Figure 4 shows that the top-N exact match and recall at N curves are almost identical. This shows that Mindstone's reader is not the performance bottleneck, and future attempts to improve the pipeline need to be focused on the retriever or ranker.

6.2 Full pipeline performance

In this section, times are measured on a machine with a single NVIDIA V100 GPU (16GB memory) and an 8-core CPU. Mixed precision[4] is used for inference. All time measurements are averaged over 1000 queries in batch mode, and the minimum of 5 runs is reported. To have a fair comparison, we have partially re-implemented DrQA to use Anserini, which improves its speed. We have implemented [23]

[4] `https://github.com/NVIDIA/apex`

Table 2: Open-domain results for SQuAD development set and Wikipedia. DrQA retrieves 5 articles, and all other systems retrieve 100 paragraphs. ↑ (↓) means higher (lower) values are better

Model	EM (↑)	F1 (↑)	Time per query (↓)
DrQA [7]	29.8	-	988 ms
BERTSerini [23]	38.6	46.1	887 ms
Yang et al. [24]	50.2	58.2	887 ms
Mindstone (ours)	**58.1**	**65.8**	738 ms

Table 3: Effect of different parts of Mindstone in open-domain SQuAD.

Model	EM (↑)	F1 (↑)	Time per query (↓)
Mindstone (ours)	**58.1**	**65.8**	738 ms
- BERT-large reader	53.9	62.7	722 ms
- ranker data augmentation	47.8	56.4	
- ranker fine-tuning	44.3	53.7	

and [24] following their descriptions in the cited papers, and use our implementation to measure time. Exact match and F1, however, are directly copied from [7], [23] and [24].

6.2.1 Open-domain QA with Wikipedia and SQuAD

Our main results for open-domain QA from Wikipedia using SQuAD questions are shown in Table 2. Our best-performing system has a BERT-large reader and a fine-tuned ranker. The ranker training approach that resulted in the best performance is fine-tuning followed by data augmentation with neural ranker (the first and third approach from section 5.1).

Although DrQA uses a much smaller neural network, its end-to-end time per query is greater than Mindstone since most of the time is spent on retriever (DrQA uses unigrams *and* bigrams, which makes the index larger and retrieval slower) and reader's preprocessing (especially named entity recognition). BERTSerini and [24] are slower than Mindstone due to the additional time spent on reader's token-level classification and processing multiple segments for documents that are longer than BERT's maximum sequence length. Mindstone on the other hand, only reads 2.5% of documents and its ranker only processes the truncated version of long paragraphs.

One of our findings is that adding a ranker enables a better use of a BERT-large reader. In previous approaches such as [23], using a BERT-large reader would increase query time by 63%, while in Mindstone it increases by only 2%. In addition, ranking is an easier task for a neural network to learn and requires less supervision. In particular, we were able to use a larger and more diverse dataset (MS MARCO) to train our ranker because ranking does not require query answers to be known exactly. In other words, we have built a pipeline in which most of the heavy-lifting is done by models trained on weaker -and cheaper to collect- labeled datasets.

We have conducted an ablation study and Table 3 shows the results. Note that even Mindstone with a BERT-base reader outperforms the previous state-of-the-art by 3.7 and 4.5 points in terms of exact match and F1.

Figure 5 shows that unlike DrQA (as shown in [15]), later stages of the pipeline do not get confused when retriever provides them with more documents. This essentially means that there is a trade-off between speed and accuracy that can be leveraged according to the timing requirements of the application.

6.2.2 Open-domain QA with Snowflake

Our results for the Snowflake dataset are in Table 4. No ranker fine-tuning is performed. In the first and the third rows, we fine-tuned BERTSerini's retriever and reader score weights on Snowflake, but not for Mindstone. Neither of the two models have been trained or further tuned on Snowflake. In the fourth row, reader is fine-tuned on Snowflake's small training set. In the second, the fifth and the sixth rows, reader is fine-tuned on the generated dataset then on Snowflake's training set.

Table 4: Open-domain results for Snowflake. All systems retrieve 50 paragraphs.

Model	EM (↑)	F1 (↑)	Time per query (↓)
BERTSerini (no fine-tuning) [23]	22.4	48.1	679 ms
+ reader fine-tuned with query generation	27.6	53.5	
Mindstone (BERT-base reader, no fine-tuning)	26.2	57.6	461 ms
+ reader fine-tuned on Snowflake	28.8	58.9	
+ reader fine-tuned with query generation	31.4	62.6	
+ BERT-large reader	**33.3**	**65.2**	470 ms

Since in Snowflake each question has only one gold answer, exact match is a harsher metric compared to SQuAD where each question has three gold answers. *Therefore, the F1 column for Snowflake results is more informative.*

We observe that *even without any additional training or fine-tuning*, Mindstone achieves great results and outperforms other models on a new dataset, and combined with our domain-adaptation approach, it results in even bigger improvements.

7 Conclusion and Future Work

This paper makes two major contributions. First, the set of techniques presented establish new state-of-the-art results for end-to-end performance on question answering for Wikipedia/SQuAD dataset (EM=58.1, F1=65.8), with an 8 points gain over previous baseline [24]). Second, the adaption techniques demonstrate large performance gains (EM and F1 gains of more than 10 points), over a system that has not been domain adapted.

While transformer-based pre-trained models have made great strides in establishing new baselines for machine comprehension for questions over passages, a large performance gap still exists between passage-level comprehension and question answering from a collection of documents. We have used a conventional retriever supplemented with a neural ranker and neural RM3 relevance feedback to boost the performance of the end-to-end pipeline. Additionally, we have demonstrated large gains in question answering performance that can be achieved using domain adaptation techniques described in the paper. Further improving the question generation component is one direction for future work.

We have focused on building highly responsive system with a sub-second latency for question answering. While conventional retrievers can operate with a small latency, the computationally heavy ranking and reader stages that use BERT, can slow down the pipeline. This has required carefully tuning of system parameters, including number of documents retrieved and ranked, and that processed by the reader. Techniques such as [10] that reduce the model size, and newer models such as ALBERT [3] that have fewer parameters can play a role in increasing the speed of QA systems.

References

[1] Data warehouse. `https://en.wikipedia.org/wiki/Data_warehouse`. Accessed: 2019-10-01.

[2] Squad leaderboard. `https://rajpurkar.github.io/SQuAD-explorer/`. Accessed: 2019-10-01.

[3] Anonymous. {ALBERT}: A lite {bert} for self-supervised learning of language representations. In *Submitted to International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=H1eA7AEtvS`. under review.

[4] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4945–4949. IEEE, 2016.

[5] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, 2013.

[6] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*, 2015.

[7] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017. doi: 10.18653/v1/p17-1171.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in neural information processing systems*, pages 1693–1701, 2015.

[10] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[11] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[12] Victor Lavrenko and W. Bruce Croft. Relevance based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, pages 120–127, New York, NY, USA, 2001. ACM. ISBN 1-58113-331-6. doi: 10.1145/383952.383972. URL `http://doi.acm.org/10.1145/383952.383972`.

[13] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human-generated machine reading comprehension dataset. 2016.

[14] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*, 2019.

[15] Martin Raison, Pierre-Emmanuel Mazaré, Rajarshi Das, and Antoine Bordes. Weaver: Deep co-encoding of questions and documents for machine reading, 2018.

[16] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[17] Siva Reddy, Danqi Chen, and Christopher D Manning. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[19] Zichao Wang, Andrew S. Lan, Weili Nie, Andrew E. Waters, Phillip J. Grimaldi, and Richard G. Baraniuk. Qg-net: A data-driven question generation model for educational content. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, L@S '18, pages 7:1–7:10, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5886-6. doi: 10.1145/3231644.3231654. URL `http://doi.acm.org/10.1145/3231644.3231654`.

[20] Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association for Computational Linguistics*, 6: 287–302, 2018.

[21] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[22] Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Enabling the use of lucene for information retrieval research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1253–1256. ACM, 2017.

[23] 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 bertserini. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 72–77, 2019.

[24] Wei Yang, Yuqing Xie, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. Data augmentation for bert fine-tuning in open-domain question answering, 2019.

[25] Yi Yang, Wen-tau Yih, and Christopher Meek. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018, 2015.

A Conversational Question Answering Interface

Mindstone allows users to interact with the system using a simple chatbot-style interface, as illustrated in Figure 6.

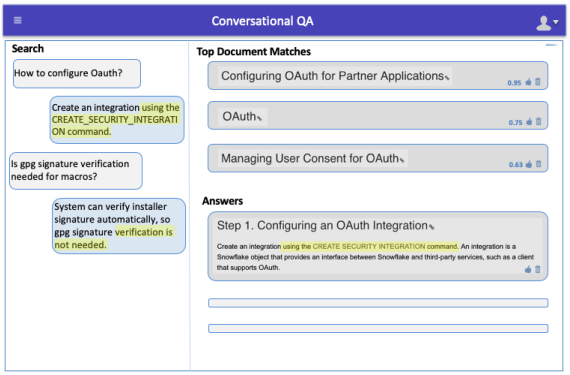

Figure 6: Mindstone conversational interface