Deep Retriever: Information Retrieval for Multi-hop Question Answering

Zijian Wang
Stanford University
zijwang@stanford.edu

Vera Xiaowen Lin
Stanford University
veralin@stanford.edu

Leo Mehr
Stanford University
leomehr@stanford.edu

Abstract

HotpotQA is a question answering dataset that features multi-hop reasoning [19]. In multi-hop QA, a question requires reasoning over multiple paragraphs in order to obtain the correct answer. One key feature of HotpotQA is that candidate paragraphs are generated from all of Wikipedia, meaning the Information Retrieval (IR) system bounds the performance of QA models. Because the 2nd-hop query term often does not appear in the question itself, basic IR systems cannot retrieve the necessary paragraphs. In this paper, we propose `Deep Retriever`, a deep learning-based IR pipeline that is designed for multi-hop retrieval with Elasticsearch. We first craft a novel dataset with heuristically-inferred labels, then build and train the Deep Retriever pipeline. Our end-to-end performance shows that Elasticsearch is a good baseline IR engine, and that Deep Retriever improves baseline EM by *24.8%* and F_1 by *15.6%*. We thus demonstrate the feasibility and promising performance of `Deep Retriever` for multi-hop QA.

1 Introduction

Recent advances in Question Answering have illustrated the capability of deep learning models to perform well on complex reasoning tasks [6, 7, 16]. However, in many QA datasets, state-of-the-art models have already surpassed human performance, indicating that the current QA datasets have much opportunity for greater complexity and difficulty [5, 15].

One major limitation of specific QA datasets such as SQuAD [16] is that the model is provided with the exact context necessary to be able to answer a given question. One very exciting research area is to develop intelligent systems that can be given a question *without any context* and still provide the correct answer. This is called **open-domain** question answering, a task that requires a combination of information retrieval and machine comprehension.

Open-domain question answering is especially interesting and challenging in the multi-hop setting because naive information retrieval is easily prone to failure. For example, consider the question:

"In which city did Mark Zuckerberg go to college?"

Basic information retrieval systems that search for documents relevant to the above question will likely find resulting documents about *Mark Zuckerberg*, Harvard, or possibly certain cities relevant to Zuckerberg. Even searching this question in Google produces an *incorrect* answer *White Plains, New York*, the city in which Zuckerberg was born. To answer this question correctly, a QA system must first identify that Zuckerberg went to *Harvard*, and then that Harvard is located in the city of *Cambridge, Massachusetts*.

In our project, we aim to improve the information retrieval step in open-domain multi-hop QA systems. We specifically work with HotpotQA, a new large-scale Wikipedia-based question answering dataset that features open-domain multi-hop reasoning [19]. The HotpotQA dataset contains 112,779

questions produced by Amazon mechanical turkers from a large subset of Wikipedia. We focus exclusively on what the papers describes as the `full-wiki` setting, in which each question is accompanied by 10 candidates paragraphs that are generated by the information retrieval system. HotpotQA's IR system uses an inverted index filtering strategy followed by choosing the 10 wiki documents that have the highest bigram tf-idf similarity with the question. The question and 10 candidate paragraphs are then concatenated and given as input to a traditional QA model.

The very important observation in this QA setting is that without the right candidate paragraphs, it is impossible for the model to produce the correct answer. Thus, the information retrieval system is a crucial bottleneck for better end-to-end performance. However, as described above, the HotpotQA IR system uses only the text from the original question to make a single search for context paragraphs. This is a clear limitation as the *Zuckerberg college* question illustrates, since *Harvard* does not exist in the question text, but needs to be identified in order to answer the question.

To solve this problem, we propose a novel information retrieval pipeline that features a neural network query generational model and Elasticsearch as a database backend. In the following sections, we present related work (§2), describe the approaches we use (§3), show initial experiment results (§4), analyze the results (§5), and conclude our work (§6).

2 Related Work

Existing QA datasets have two substantial shortcomings: they fail to perform complex multi-step reasoning and provide explanations for answers. HotPotQA [19] is a novel dataset that addresses these two shortcomings: (1) the questions require multiple supporting documents to answer and (2) the dataset provides sentence-level supporting facts required for obtaining an answer. For comparison, SQuAD [16] does not provide (1) – the necessity of using multiple documents to obtain an answer, also known as *multi-hop* reasoning. In SQuAD, not only is a single document needed, but the answer is often found by matching a single sentence within the document. Further, once an answer is found in a SQuAD task, there is very little opportunity for explainability. HotPotQA labels the sentences that are supporting facts for its examples, allowing a QA system to learn with strong supervision and provide explanations for its predictions.

Open-domain QA is defined as finding answers in collections of unstructured documents. Open-domain QA using Wikipedia is particularly challenging due to the large scale of the knowledge source, which involves machine reading at scale. DrQA tackles this problem by breaking it down into two parts, document-retrieval and machine comprehension, where the first component, Document Retriever, finds relevant articles and the second component, Document Reader, extracts answers from the retrieved articles [2]. The Document Reader is a machine comprehension model similar to the Attentive Reader [1], but adds in features cleverly crafted for the task, including part-of-speech (POS), named entity recognition (NER) tags and its (normalized) term frequency (TF), which yields better performance. Although single-hop open-domain QA has been explored previously [2], the multi-hop open-domain QA remains relatively open.

One of the most important components in QA systems is "information retrieval" (IR). IR is defined as finding documents of an unstructured nature (usually text) that satisfies an information need from within large collections [11]. Among techniques in the IR field, tf–idf, short for "term frequency–inverse document frequency", which measures how important a word is to a document in a collection or corpus by multiplying word term frequency and its inverse document frequency, has been widely used as the weighting factor in searches of information retrieval [10]. This technique was also applied in HotpotQA [19], however, it is not the best scheme due to HotpotQA's multi-hop property, as the second-hop term may not appear in the query itself. More recent work using deep learning for information retrieval (e.g., [13], [4]), but most of them suffer from problems either is not designed for the multi-hop QA or are inefficient to generalize easily to full wikipedia dataset.

3 Approaches

3.1 Overview

We present Deep Retriever: a *novel* neural network-based pipeline to retrieve relevant information for multi-hop question answering. The input to Deep Retriever is a multi-hop question and output is a

list of paragraphs that contain information sufficient for answering it. For a visual description of this pipeline, please examine Figure 1. The pipeline consists of the following components:

- Query 1 Generator: produce the 1st query solely from the question text (example: *Mark Zuckerberg*).
- Search 1: using the query produced by query 1 generator, search Elasticsearch for the first 5 context paragraphs (example: top 5 paragraphs related to *Zuckerberg*).
- Query 2 Generator: use the question and the first 5 context paragraphs to generate the 2nd query term (example: *Harvard*).
- Search 2: using the 2nd query term, search Elasticsearch for the second 5 context paragraphs (example: top 5 paragraphs related to *Harvard*).
- Output: concatenate the two sets of 5 paragraphs to produces the 10 final context paragraphs.

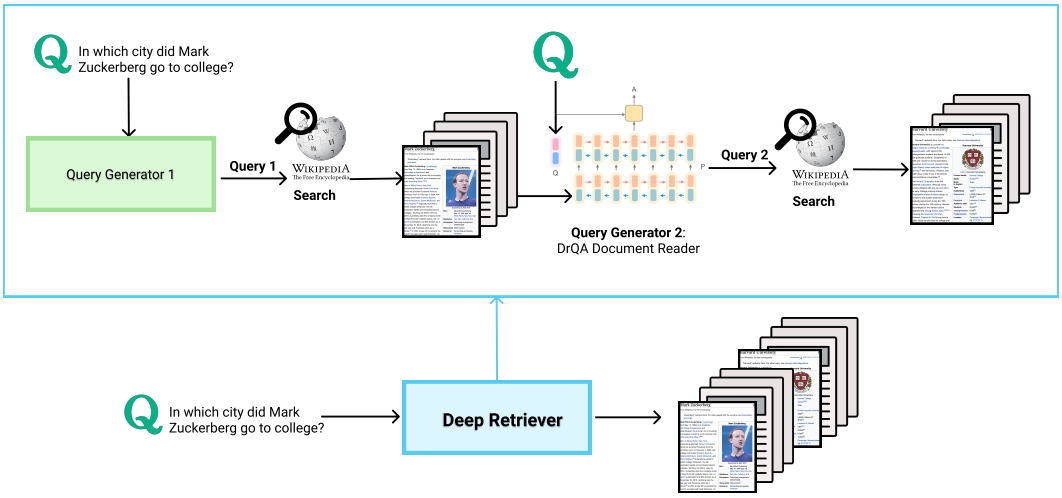

Figure 1: Deep Retriever Architecture Overview

3.2 Elasticsearch

We use Elasticsearch (ES) as our search engine to retrieve Wikipedia documents. ES is a very powerful system that provides many out-of-the-box text search features and is commonly employed in real-world IR systems (such as Wikipedia itself, Stack Overflow, Github, and many more). ES scores documents using tf-idf and field-length norm[1].

We implemented the enrity of our data processing, indexing, and querying code using the `elasticsearch-py`[2] library. We run ES as a server on an Azure machine more powerful than our laptops: Standard D16s v3, 16 vcpus, 64 GB memory. Our pipeline configures ES to store our data; extracts the Wikipedia corpus from the HotPotQA training data and inserts it into ES; and provides an API for our IR system to easily query ES.

3.3 Query 1 Generator

The Query 1 Generator takes the question as input and produces a subspan or subsequence that represents the best string to provide to Query 1 (e.g. *Mark Zuckerberg*). Based on the dataset we created (described in §4.1 and §4.2), we implemented 2 deep neural net architectures ourselves to model this task: (1) the Sequence-to-sequence (seq2seq) architecture [18], and (2) long short-term memory (LSTM, [8]) with pre-trained embeddings [14] to extract a span the original question. However, our seq2seq implementation was unsuccessful in producing valid output, often failing to generate an end token and spouting many sentences of related gibberish. We originally chose the seq2seq because of its ability to generate non-continuous subsequences of the input question as well

[1] `https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html`
[2] `https://github.com/elastic/elasticsearch-py`

as tokens that were not in the input. However, after some more careful analysis of the dataset and discussion with our mentor, we attempted a second model: LSTMs with pre-trained embeddings. However, as initial results for this model appeared modest, we tried a third and final approach: the search ES directly with the entire question text itself. Ultimately, we found that as the target first query term almost always appears directly in the question and that ES performed quite well. Thus, our Query Generator 1 is indeed an identity operation: the output is the identical to the input. In Appendix A, we provide lengthy explanation and analysis of our first two attempted models.

3.4 Query 2 Generator

For the second generation model, we adopt the same architecture as Document Reader in DrQA [2]. We implement a multi-layer bidirectional LSTM with dropout [3]. Following the terminology in [2], we formulate the problem as given a small set of documents of n paragraphs and a question q, find the answer: the n paragraphs is the documents retrieved by ES using the first query, the question is the original question, and the answer is the desired second search query, a span of tokens from the paragraphs that are likely to find the gold paragraphs in Elasticsearch.

3.5 Multi-hop QA Model

In order to test end-to-end performance in our open-domain multi-hop QA task, we feed to the output of Deep Retriever to a multi-hop QA model. We use the same model architecture as introduced in the HotpotQA paper, which itself is adopted from [3] and features state-of-the-art technical advances like character-level models, self-attention, and bi-attention [19].

4 Experiments

4.1 Dataset Generation

We use the HotpotQA dataset, which provides a training, dev, and test split. However, because the provided test set is not labelled (i.e. answers and supporting facts), we split the dev set 50/50 into our own dev and test sets (henceforth, `dev` and `test`). The training set contains 90,447 questions and both dev and test sets contain 3,702 questions. Our Wikipedia corpus contains 482,021 paragraphs.

We split our train data further in order to avoid data leakage and overfitting to the dev set while training our Query Generators. We split `train` three ways: 80% train, 10% dev, and 10% test (henceforth, `train-train`, `train-dev`, and `train-test`).

In our dataset, each question is labelled with two **gold paragraph titles**. These are the titles of the two Wikipedia paragraphs that contain exactly enough information to answer the corresponding question. Henceforth, we refer to these interchangeably as the `gold paragraphs` or `gold titles`.

4.2 Label Generation

We infer our labels (`label-1` for Query 1 Generator and `label-2` for Query 2 Generator) for the dataset from the titles of the gold paragraphs. To do this, we first tokenize and preprocess all questions and corresponding titles of the two gold paragraphs via Stanford CoreNLP 3.9.2 [12]. Then, we look at whether there are exact matches between the title and each question. If so, the one matched will be `label-1` and the other will be `label-2`; and if both matched, the one occurs earlier will be `label-1` and the other one will be `label-2`. If there is no exact match, we look at the longest common substrings (LCS) for each title in the question, and the one with high percentage of existed words ($len(LCS)/len(title)$) will be `label-1`. If there is no LCS in both titles, as the query to be searched should normally be a Noun Phrase (NP) with some infrequent words like name, location, and nationality, we generate our own `label-1` heuristically through taking the longest NP with the lowest term frequency [17] in the question as `label-1` and concatenate the two titles as `label-2`.

4.3 Evaluation methods

We evaluate each important step of our model pipeline:

[3] References: `https://github.com/facebookresearch/DrQA`

- Query 1: Distribution of gold paragraph ranking. See §4.5.1
- Query Generator 2: EM (Exact Match) and F_1. See §4.5.2
- Overall Deep Retriever output: Hits@10. See §4.5.3
- End-to-end model: EM and F_1 scores with the answer, the supporting fact, and a joint value between these two [19]. See §4.5.4

Hit@n evaluates the performance of the IR system. The definition of Hit@n is

$$\text{Hit@}n \quad = \quad \frac{|\{\text{relevant documents in top } n \text{ retrieved results}\}|}{|\{\text{relevant documents}\}|}$$

As [19] uses Hit@10 in evaluation, we adopt the same metrics for our IR systems.

In terms of the end-to-end evaluation, we also adopt metrics (Joint F_1 and EM) in [19], which are:

$$\text{P(joint)} = \text{P(ans)} * \text{P(sup)}, \text{R(joint)} = \text{R(ans)} * \text{R(sup)}$$

$$\text{Joint } F_1 \quad = \quad \frac{2 * \text{P(joint)} * \text{R(joint)}}{\text{P(joint)} + \text{R(joint)}}$$

where P and R stands for precision and recall, and Joint EM is 1 only if both tasks achieve an exact match and otherwise 0 [19].

4.4 Experimental details

For the Query 2 Generator model, we use an RNN model with pre-trained word embeddings and pre-trained DrQA model for a warm start. The loss function is negative log likelihood. We use mini-batch gradient descent, along with an adam optimizer to train our model. Details on model parameters can be found in Appendix B.

As of HotpotQA model, since the model was not publicly released, we train our models use the same parameter setting as recommended[4]. Using the same settings, we train models with the original splitted dataset in [19] and our generated dataset from our retrieval system.

4.5 Results

We split the presentation of our results over several subsections, corresponding to different parts of the Deep Retriever pipeline, as well as to the end-to-end performance.

4.5.1 Query 1

Query 1 returns 5 paragraphs that are then fed to the Query 2 Generator. In order to evaluate how well we are retrieving relevant paragraphs, we instead have Query 1 return 10 paragraphs and examine the position of the first gold paragraph in the results. Elasticsearch orders its results by relevancy, so we expect that the first paragraphs are more likely to contain the gold paragraph.

Figure 2 illustrates the cumulative percentage of queries in which the results contain the first gold paragraph before the ith position in the result list. For example, for nearly 50% of the questions in our dataset, the first paragraph returned by Elasticsearch is in fact the gold paragraph. We see that the marginal benefit of returning more paragraphs from the first query decreases. We chose 5 paragraphs as our cutoff because we need to allow space for the Query 2 results (since Deep Retriever must return 10 paragraphs total, as in our specification), and because with 5 we already capture the first gold paragraph for 75% of the questions.

[4] See "Training" section in `https://github.com/hotpotqa/hotpot`

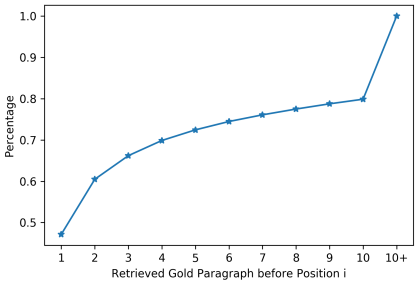

Figure 2: Cumulative plot of the first hit position of the first gold paragraph when searching directly

4.5.2 Query 2 Generator Performance

Split	Answer-EM	Answer-F_1
train-dev	61.43	67.22
train-test	60.30	65.86

Table 1: Performance comparisons in train

Split	Answer-EM	Answer-F_1
dev	47.63	53.54
test	47.30	52.95

Table 2: Performance comparisons in dev

We can see that in both Table 1 and Table 2, the different between the splits are negligible. However, there is a relatively big gap between the performance of in Table 1 and Table 2. `train-dev` is much higher than `dev-dev` and `train-test` is much higher than `dev-test`. This gap is expected, because the HotpotQA train/dev split is not actually homogeneous. The train dataset contains many questions with medium difficulty while the dev dataset contains only hard questions [19]. Recall that as discussed in §4.2, the labels for query 2 are extracted with heuristiscs so an exact match to the label although likely, does not guarantee finding the gold paragraphs. On the other hand, through closer inspection, we notice that some predictions are either a substring of the label or contain the label (i.e., a superstring), which could retrieve the correct gold paragraphs in both cases.

4.5.3 Deep Retriever Performance

We examine the Hits@10 ratio for the 10 paragraphs outputted by Deep Retriever and compare it to both the HotpotQA IR system and our Elasticsearch baseline in Table 3. Unsurprisingly, our baseline results are very similar to the HotpotQA paper as ES uses tf-idf as well as some basic grammatical tokenization, strategies that were also employed in the HotpotQA IR system. Our Deep Retriever performs better than our baseline, but we were very surprised to find that it did not in fact perform better than the HotPotQA IR engine. Further, we were surprised that our novel system performed only slightly better than the baseline. However, there are several reasons for error to propagate that we observed. As we saw earlier, Query 1 only captures around 75% of the gold paragraphs, which limits the possible performance of Query 2 Generator. Further, Query 2 Generator has an EM of around 50% for capturing the second paragraph. Ultimately, our Hit@10 ratio should roughly bounded by the average of these two.

However, as we show in §5, Hits@10 can be an underestimate of IR performance, which we believe is ultimately the cause of our end-to-end performance being greater using Deep Retriever.

	Split	HotpotQA	Baseline ES	Deep Retriever
Hits@10	dev	56.06	49.55	52.94
	test	55.88	48.29	51.65

Table 3: Information retrieval performance comparisons

4.5.4 End-to-end Performance

We train separate models using the dataset with the IR system in [19] our ES baseline, and our retrieval pipeline[5]. Adopting the evaluation metrics in [19], we compare the performances in Table 4.

From the results, we see that our baseline retrieval system (middle ones) already performs closely, if not better than the bigram tf-idf based retrieval system in [19]. This validates the feasibility of using Elasticsearch as the baseline IR system.

Retrieval System	Split	Answer		Sup Fact		Joint	
		EM	F_1	EM	F_1	EM	F_1
HotpotQA	dev	25.14	34.63	4.97	37.04	2.59	16.88
	test	22.95	32.44	3.83	35.84	1.73	15.09
Baseline ES	dev	25.12	34.67	5.97	36.37	2.84	18.26
	test	23.04	32.07	5.64	35.26	2.78	16.75
Deep Retriever	dev	27.99	37.67	6.64	38.79	3.68	20.80
	test	**26.23**	**35.68**	**6.50**	**36.98**	**3.47**	**19.37**

Table 4: End-to-end performance comparison in `full wiki` setting. (cf. Table 4 in [19])

Further, we achieve better performances using our multi-hop retrieval pipeline: the joint EM achieves an 100.5% increase over the baseline in [19] and 24.8% over ES baseline; while the joint F_1 achieves 28.4% and 15.6%, respectively. Those results show that our multi-hop retrieval system could help improve the performance of multi-hop question answering system significantly.

5 Analysis

5.1 Error Analysis for Query 2 Generator

Several examples are presented in Appendix C. On a high level, some problems are due to the constrains of the current model architecture, some are caused by the limitation of the original HotpotQA datasets, and some are introduced during the label generation process.

Appendix C.1 shows that the model sometimes predicts query 1 twice. In comparison model, usually both of the search terms are contained in the original question but the query 2 is often not in the query 1 results. When given the question and paragraphs, it would be difficult for the model to know which search term to pick, resulting in duplicate queries.

Appendix C.2 is an example of ambiguous search term. Although the model correctly captures the desired second query, sometimes the search term is overloaded with many meanings. The search engine will likely return many relevant results but has no insight on which one we are referring to. This problem could potentially be solved by adding context to the search term, which requires a subsequence instead of substring of word tokens.

There are another type of questions which suffers from similar issue. When the Turkers create the questions for HotpotQA dataset, they are presented with some relevant paragraphs. Sometimes they use only first name or last name to refer a person, which is clear with the relevant paragraphs present, but becomes ambiguous without the context. This ambiguity will also confuse our search engine.

Appendix C.3 represents a group of errors introduced during the label generation process. Since the original HotpotQA dataset does not include the type of the question or the ground truth of search queries, we have to come up with some heuristics to generate our labels as descried in §4.2. These heuristics are not perfect and can lead to incorrect label. For example, in Appendix C.3, when there is no exact match to the ideal regex pattern in the paragraphs and the backup fuzzy match returns a very long span. This particular long label error only appears in around 0.3% of the training data so we simply drop them in the training set.

[5]To avoid data leakage, the training dataset is the same with the baseline ES and not using the pipeline.

5.2 Underestimation of Hits Ratio

We observed cases were although our queries do not produce the correct gold paragraphs but are able to produce other paragraphs that still contain the desired information. That is, there are cases where our model does not retrieve a gold paragraph, but can still find the information necessary for answering the question.

One example is the question: *Carrie Ann Inaba was a judge for a show that was featured on what television network?* For this question, `label-1` is *Carrie Ann Inaba*, `label-2` is *Dancing with the Stars U.S. season 13*, and the answer is *ABC TV*. The output of our Query 1 produced paragraphs titled *Carrie Ann Inaba*, *Dancing with the Stars (U.S. season 22)*, and 3 others. Note that our retrieval was able to pick out a similar article to the 2nd hop gold paragraph (*Dancing with the Stars*), but instead of *season 13*, found *season 22*. Looking at the text the of the *season 22* paragraph, we see that it clearly indicates that the show was featured on *ABC TV*. This example illustrates it is possible for our IR system to miss a gold paragraph (i.e. decreasing the hit ratio), but still find the important information that could allow it to answer the final question (thus boosting final answer EM).

5.3 Analysis for End-to-end Pipeline

With *Deep Retriever*, the QA model is able to answer more questions correctly as it is provided with better context. For example, given the question *The television series in which Ekaterina Klimova played Dutchess Natalia Repnina took place in what century*, our model is able to pick the queries (*Ekaterina Klimova* and *Poor Nastya*) correctly and gives the correct answer *the 19th century*, while with the baseline IR system, it is not possible to get the second query and thus give the wrong answer.

However, the end-to-end performance could be affected by the error propagated from the very first label inference, to elasticsearch retrieval results and query generations. All these factors may lead to insufficient context provided into the end-to-end model and lead to worse performances.

The hyperparameters of the model was not fine-tuned per dataset. The default hyperparameters may be a good start for the original dataset, however, with Elasticsearch and the new pipeline, it may not work well. Thus, there is still potential to improve the performance via fine-tuning the model more.

5.4 Future Work

Based on our results and analysis, we highlight a few future work that could potentially make the result robust:

- Develop better query 1 generator by trying out heuristic ways (e.g., with named entity recognition/NER) and deep learning models as mentioned in §3.3.
- Fine-tune the distribution of the number of paragraphs returned by Search 1 and Search 2. We decided on a 5-5 split for the first and second query (see discussion for Figure 2), but could run experiments to determine whether a different distribution would perform better.
- Investigate comprehensive methods to evaluate gold paragraphs and supporting facts, e.g., using sentence/document similarity or crowdsourcing more variations of gold paragraphs.

6 Conclusion

Traditional IR search engines suffer in the multi-hop QA setting as they may not be able to find information necessary to make the 2nd hop. In this paper, we propose `Deep Retriever`, a deep learning-based IR pipeline that is designed for multi-hop retrieval with Elasticsearch. Using the multi-hop QA dataset HotpotQA, we investigate methods to alleviate the multi-hop retrieval issue. Our main contributions include: 1) creating a *new* dataset with heuristically-inferred labels for query generation tasks, 2) building a deep-learning based pipeline to perform multi-hop retrievals, 3) demonstrating that our pipeline considerably increases EM by *24.8%* and F_1 by *15.6%* over the existing baselines, and 4) providing further analysis on results and next steps for making multi-hop retrieval systems even more powerful. Our work fills a key missing gap of the suitable IR system for multi-hop question answering.

Acknowledgement

We thank our mentor Peng Qi and our in-class mentor Amita Kamath for their great help on this project.

References

[1] Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. *CoRR*, abs/1606.02858, 2016.

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

[3] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. *CoRR*, abs/1710.10723, 2017.

[4] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, and Andrew McCallum. Multi-step retriever-reader interaction for scalable open-domain question answering. In *International Conference on Learning Representations*, 2019.

[5] 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.

[6] 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.

[7] Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301*, 2015.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge university press, 2014.

[11] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

[12] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[13] Bhaskar Mitra and Nick Craswell. Neural models for information retrieval. *arXiv preprint arXiv:1705.01509*, 2017.

[14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[15] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 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] Robyn Speer, Joshua Chin, Andrew Lin, Sara Jewett, and Lance Nathan. Luminosoinsight/wordfreq: v2.2, October 2018.

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[19] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Appendices

A Query 1 Generator Models

A.1 seq2seq Model

This was our first version of Query 1 Generator. We used a seq2seq model to generate the first query. We tried two sets of pre-trained GloVe word embeddings: the 100-dimensional pre-trained embedding `glove.6B.100d` and the 300-dimensional pre-trained embedding `glove.840B.300d` [14]. We froze the embeddings for `glove.840B.300d` but left the embeddings trainable for `glove.6B.100d`.

For training, we use mini-batch stochastic gradient descent to minimize negative log-likelihood. We use a the learning rate `lr = 0.001` with an exponential decay at the rate of `decay_rate = 0.9999`. Training is performed with shuffled mini-batches of size 32. We set hyperparameters as `enc_hidden_size = 150`, `dec_hidden_size = 200`, `enc_rnn_dropout = 0.2`, `dec_rnn_dropout = 0.2`.

Adam optimizer [9] was used to speed up the training. Due to the small training set of less than $100,000$ (question, span) pairs, the model converges rather quickly in under two hours on a CPU machine. We also tried various combinations like `dropout = 0.4`, `batch_size = 128`, `enc_rnn_dropout = 0.4`, `dec_rnn_dropout = 0.4`, etc.

Since the query generator was trained on the heuristic search span data instead of the target search queries, here we present a qualitative analysis of the query generator performance. Some sample outputs are shown below:

Example 1
Question: Cadmium Chloride is slightly soluble in this chemical, it is also called what??
Target search query: Cadmium Chloride
seq2seq query generator: Cadmium Chloride

Example 2
Question: The Oberoi family is part of a hotel company that has a head office in what city?
Target search query: Oberoi family
seq2seq query generator: The Oberoi family The The family family family a The family of a Oberoi (film)

Example 3
Question: What nationality was James Henry Miller's wife?
Target search query: Peggy Seeger
Seq2seq query generator: James Henry Henry Miller's (film)

Example 1 is an example of when the model successfully predicts the target search term. Example 2 is a common mistake the model makes: it correctly predicts the search term in the first few words but it fails to predict an <EOD> at the end of the search term. Example 3 presents an interesting case where the heuristic search span is wrong (it should be the second search term instead of the first) but our model was able to generalize and finds the correct search term. Although it suffers from an issue similar to the one in Example 2, it demonstrates our model's ability to generalize.

For the common error shown in Example 2, we identify several possible causes: (1) after predicting the last word in the search term, the decoder incorrectly predicts the next word and then the error propagates; (2) some of the query are uncommon names which do not have good pre-trained embeddings; and (3) the loss function does not penalize the length difference in the output harsh enough. Since seq2seq is notoriously bad at predicting the end token, we decided to switch to another model architecture.

A.2 Bi-LSTM Model

As the seq2seq model is too complcated and suffers problems mentioned above, we opted to try a simpler Bidirectional LSTM model for the first query generation. Specifically, the model was designed to be many-to-many, where each word has a binary label - to be included in the query (1) or not (0). It use the pretrained GloVe embedding `glove.840B.300d`, with an auxiliary Named Entity

Recognition (NER, inferred using Stanford CoreNLP [12]) embedding as the target query has a high chance to fall in those special categories (e.g., organizations, names, and locations). The LSTM is two-layer stacked with `hidden size = 300`. We use `BCELoss`[6] with a mask layer to deal with padding issue and a weight layer to deal with data imbalance issue, as there are only few positive labels in the question. The model was trained with Adam optimizer [9] with `batch_size = 64` and `droupout = 0.4`. As we care more about whether the model is able to pick the correct terms, we evaluate the model using two metrics: recall and F_1 score.

However, the Bi-LSTM model does not seem to overfit to the training dataset well, even with 0 dropout rate. This suggests that the task of predicting each word into binary labels may be too difficult. The next possible way to try is to change the objective to be predicting the start position of the selected span, and conditioning on that to predict the end position of the span, which is a similar architecture compared to the top layers of the model in HotpotQA [19].

B Query 2 Generator Model Parameters

```
-embedding-file glove.840B.300d.txt
-tune-partial 100
-pretrained data/reader/single.mdl
-max-len 10
-expand-dictionary True
-official-eval False
-valid-metric exact match
-num-epochs 15
-dropout-rnn 0.4
-dropout-emb 0.4
-learning-rate 0.1
-grad-clipping 10
-tune-partial 1000
```

Documentations about the parameters can be found at the DrQA Github page[7].

C Query 2 Generator Error Examples

C.1 Comparison Question

Question: Who has had more names, University of Texas at Austin or University of Greenwich??
Label: University of Texas at Austin
Prediction: University of Greenwich

C.2 Ambiguous Entry

Question: Stuart Besser is a film producer that has appeared as an actor in a film directed by who?
Label: Identity
Prediction: Identity

C.3 Long Label

Longest common substring: Cavendish Marquess of Hartington
Regex pattern: Cavendish Marquess of Hartington
Backup regex pattern:: Cavendish.*?Marquess.*?of.*?Hartington
Matched span:: Cavendish, 10th Duke of Devonshire, and his wife, Lady Mary Gascoyne-Cecil. He was the husband of Kathleen Kennedy, sister of the future U.S. President John F. Kennedy. Kathleen Agnes Cavendish, Marchioness of Hartington ("née" Kennedy; February 20, 1920 – May 13, 1948), also known as "Kick" Kennedy, was an American socialite. She was the daughter of Joseph P.

[6] `https://pytorch.org/docs/stable/nn.html#torch.nn.BCELoss`
[7] `https://github.com/facebookresearch/DrQA/blob/master/scripts/reader/README.md`

Kennedy, Sr. and Rose Kennedy, sister of future U.S. President John F. Kennedy, and wife of the Marquess of Hartington