Answering Complex Open-domain Questions Through Iterative Query Generation

Peng Qi† **Xiaowen Lin***† **Leo Mehr***† **Zijian Wang***‡ **Christopher D. Manning**†
† Computer Science Department ‡ Symbolic Systems Program
Stanford University
{pengqi, veralin, leomehr, zijwang, manning}@cs.stanford.edu

Abstract

It is challenging for current one-step retrieve-and-read question answering (QA) systems to answer questions like *"Which novel by the author of 'Armada' will be adapted as a feature film by Steven Spielberg?"* because the question seldom contains retrievable clues about the missing entity (here, the author). Answering such a question requires multi-hop reasoning where one must gather information about the missing entity (or facts) to proceed with further reasoning. We present GOLDEN (Gold Entity) Retriever, which iterates between reading context and retrieving more supporting documents to answer open-domain multi-hop questions. Instead of using opaque and computationally expensive neural retrieval models, GOLDEN Retriever generates natural language search queries given the question and available context, and leverages off-the-shelf information retrieval systems to query for missing entities. This allows GOLDEN Retriever to scale up efficiently for open-domain multi-hop reasoning while maintaining interpretability. We evaluate GOLDEN Retriever on the recently proposed open-domain multi-hop QA dataset, HOTPOTQA, and demonstrate that it outperforms the best previously published model despite not using pretrained language models such as BERT.

1 Introduction

Open-domain question answering (QA) is an important means for us to make use of knowledge in large text corpora and enables diverse queries without requiring a knowledge schema ahead of time. Enabling such systems to perform multi-step inference can further expand our capability to explore the knowledge in these corpora (*e.g.*, see Figure 1).

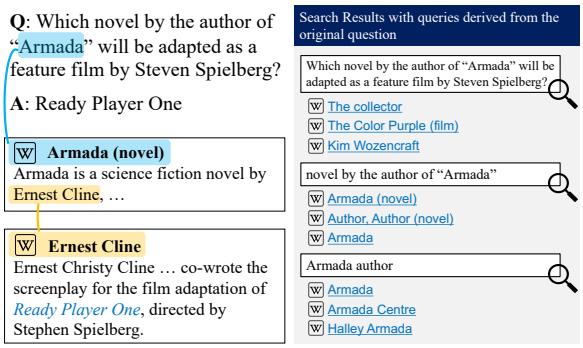

Figure 1: An example of an open-domain multi-hop question from the HOTPOTQA dev set, where "Ernest Cline" is the missing entity. Note from the search results that it cannot be easily retrieved based on merely the question. (Best viewed in color)

Fueled by the recently proposed large-scale QA datasets such as SQuAD (Rajpurkar et al., 2016, 2018) and TriviaQA (Joshi et al., 2017), much progress has been made in open-domain question answering. Chen et al. (2017) proposed a two-stage approach of retrieving relevant content with the question, then reading the paragraphs returned by the information retrieval (IR) component to arrive at the final answer. This "*retrieve and read*" approach has since been adopted and extended in various open-domain QA systems (Nishida et al., 2018; Kratzwald and Feuerriegel, 2018), but it is inherently limited to answering questions that do not require multi-hop/multi-step reasoning. This is because for many multi-hop questions, not all the relevant context can be obtained in a single retrieval step (*e.g.*, "Ernest Cline" in Figure 1).

More recently, the emergence of multi-hop question answering datasets such as QAngaroo (Welbl et al., 2018) and HOTPOTQA (Yang et al., 2018) has sparked interest in multi-hop QA in the research community. Designed to be more challenging than SQuAD-like datasets, they feature questions that require context of more than one

*Equal contribution, order decided by a random number generator.

document to answer, testing QA systems' abilities to infer the answer in the presence of multiple pieces of evidence and to efficiently find the evidence in a large pool of candidate documents. However, since these datasets are still relatively new, most of the existing research focuses on the few-document setting where a relatively small set of context documents is given, which is guaranteed to contain the "gold" context documents, all those from which the answer comes (De Cao et al., 2019; Zhong et al., 2019).

In this paper, we present GOLDEN (Gold Entity) Retriever. Rather than relying purely on the original question to retrieve passages, the central innovation is that at each step the model also uses IR results from previous hops of reasoning to generate a new natural language query and retrieve new evidence to answer the original question. For the example in Figure 1, GOLDEN would first generate a query to retrieve *Armada (novel)* based on the question, then query for *Ernest Cline* based on newly gained knowledge in that article. This allows GOLDEN to leverage off-the-shelf, general-purpose IR systems to scale open-domain multi-hop reasoning to millions of documents efficiently, and to do so in an interpretable manner. Combined with a QA module that extends BiDAF++ (Clark and Gardner, 2017), our final system outperforms the best previously published system on the open-domain (fullwiki) setting of HOTPOTQA without using powerful pretrained language models like BERT (Devlin et al., 2019).

The main contributions of this paper are: (a) a novel iterative retrieve-and-read framework capable of multi-hop reasoning in open-domain QA;[1] (b) a natural language query generation approach that guarantees interpretability in the multi-hop evidence gathering process; (c) an efficient training procedure to enable query generation with minimal supervision signal that significantly boosts recall of gold supporting documents in retrieval.

2 Related Work

Open-domain question answering (QA) Inspired by the series of TREC QA competitions,[2] Chen et al. (2017) were among the first to adapt neural QA models to the open-domain setting. They built a simple inverted index lookup with TF-IDF on the English Wikipedia, and used the question as the query to retrieve top 5 results for a reader model to produce answers with. Recent work on open-domain question answering largely follow this retrieve-and-read approach, and focus on improving the information retrieval component with question answering performance in consideration (Nishida et al., 2018; Kratzwald and Feuerriegel, 2018; Nogueira et al., 2019). However, these one-step retrieve-and-read approaches are fundamentally ill-equipped to address questions that require multi-hop reasoning, especially when necessary evidence is not readily retrievable with the question.

Multi-hop QA datasets QAngaroo (Welbl et al., 2018) and HOTPOTQA (Yang et al., 2018) are among the largest-scale multi-hop QA datasets to date. While the former is constructed around a knowledge base and the knowledge schema therein, the latter adopts a free-form question generation process in crowdsourcing and span-based evaluation. Both datasets feature a few-document setting where the gold supporting facts are provided along with a small set of distractors to ease the computational burden. However, researchers have shown that this sometimes results in gameable contexts, and thus does not always test the model's capability of multi-hop reasoning (Chen and Durrett, 2019; Min et al., 2019a). Therefore, in this work, we focus on the fullwiki setting of HOTPOTQA, which features a truly open-domain setting with more diverse questions.

Multi-hop QA systems At a broader level, the need for multi-step searches, query task decomposition, and subtask extraction has been clearly recognized in the IR community (Hassan Awadallah et al., 2014; Mehrotra et al., 2016; Mehrotra and Yilmaz, 2017), but multi-hop QA has only recently been studied closely with the release of large-scale datasets. Much research has focused on enabling multi-hop reasoning in question answering models in the few-document setting, *e.g.*, by modeling entity graphs (De Cao et al., 2019) or scoring answer candidates against the context (Zhong et al., 2019). These approaches, however, suffer from scalability issues when the number of supporting documents and/or answer candidates grow beyond a few dozen. Ding et al. (2019) apply entity graph modeling to HOTPOTQA, where

[1] Code and pretrained models available at https://github.com/qipeng/golden-retriever

[2] http://trec.nist.gov/data/qamain.html

they expand a small entity graph starting from the question to arrive at the context for the QA model. However, centered around entity names, this model risks missing purely descriptive clues in the question. Das et al. (2019) propose a neural retriever trained with distant supervision to bias towards paragraphs containing answers to the given questions, which is then used in a multi-step reader-reasoner framework. This does not fundamentally address the discoverability issue in open-domain multi-hop QA, however, because usually not all the evidence can be directly retrieved with the question. Besides, the neural retrieval model lacks explainability, which is crucial in real-world applications. Talmor and Berant (2018), instead, propose to answer multi-hop questions at scale by decomposing the question into sub-questions and perform iterative retrieval and question answering, which shares very similar motivations as our work. However, the questions studied in that work are based on logical forms of a fixed schema, which yields additional supervision for question decomposition but limits the diversity of questions. More recently, Min et al. (2019b) apply a similar idea to HotpotQA, but this approach similarly requires additional annotations for decomposition, and the authors did not apply it to iterative retrieval.

3 Model

In this section, we formally define the problem of open-domain multi-hop question answering, and motivate the architecture of the proposed GoldEn (Gold Entity) Retriever model. We then detail the query generation components as well as how to derive supervision signal for them, before concluding with the QA component.

3.1 Problem Statement

We define the problem of *open-domain multi-hop QA* as one involving a question q, and S relevant (gold) supporting context documents $d_1, \ldots, d_S$ which contain the desired answer a.[3] These supporting documents form a chain of reasoning necessary to arrive at the answer, and they come from a large corpus of documents $\mathcal{D}$ where $|\mathcal{D}| \gg S$. In this chain of reasoning, the supporting documents are usually connected via shared entities or textual similarities (*e.g.*, they describe similar entities or events), but these connections do not necessarily conform to any predefined knowledge schema.

We contrast this to what we call the *few-document setting* of multi-hop QA, where the QA system is presented with a small set of documents $\mathcal{D}_{\text{few-doc}} = \{d_1, \ldots, d_S, d'_1, \ldots, d'_D\}$, where $d'_1, \ldots, d'_D$ comprise a small set of "distractor" documents that test whether the system is able to pick out the correct set of supporting documents in the presence of noise. This setting is suitable for testing QA systems' ability to perform multi-hop reasoning given the gold supporting documents with bounded computational budget, but we argue that it is far from a realistic one. In practice, an open-domain QA system has to locate all gold supporting documents from $\mathcal{D}$ on its own, and as shown in Figure 1, this is often difficult for multi-hop questions based on the original question alone, as not all gold documents are easily retrievable given the question.

To address this gold context discoverability issue, we argue that it is necessary to move away from a single-hop retrieve-and-read approach where the original question serves as the search query. In the next section, we introduce GoldEn Retriever, which addresses this problem by iterating between retrieving more documents and reading the context for multiple rounds.

3.2 Model Overview

Essentially, the challenge of open-domain multi-hop QA lies in the fact that the information need of the user $(q \rightarrow a)$ cannot be readily satisfied by any information retrieval (IR) system that models merely the similarity between the question q and the documents. This is because the true information need will only unfold with progressive reasoning and discovery of supporting facts. Therefore, one cannot rely solely on a similarity-based IR system for such iterative reasoning, because the potential pool of relevant documents grows exponentially with the number of hops of reasoning.

To this end, we propose GoldEn (Gold Entity) Retriever, which makes use of the gold document[4] information available in the QA dataset at training time to iteratively query for more relevant supporting documents during each hop of reasoning. Instead of relying on the original question as the search query to retrieve all supporting facts,

[3] In this work, we only consider extractive, or span-based, QA tasks, but the problem statement and the proposed method apply to generative QA tasks as well.

[4] In HotpotQA, documents usually describe entities, thus we use "documents" and "entities" interchangeably.

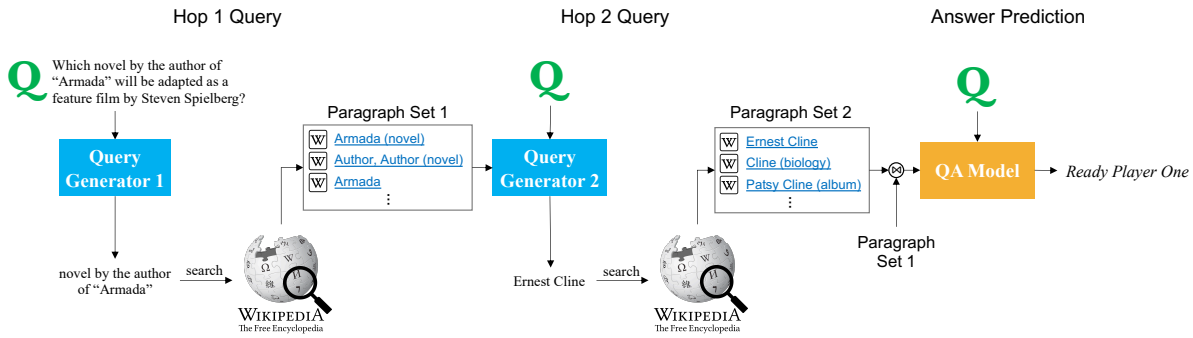

Figure 2: Model overview of GOLDEN Retriever. Given an open-domain multi-hop question, the model iteratively retrieves more context documents, and concatenates all retrieved context for a QA model to answer from.

or building computationally expensive search engines that are less interpretable to humans, we propose to leverage text-based IR engines for interpretability, and generate different search queries as each reasoning step unfolds. In the very first hop of reasoning, GOLDEN Retriever is presented the original question q, from which it generates a search query q_1 that retrieves supporting document d_1.[5] Then for each of the subsequent reasoning steps ($k = 2, \ldots, S$), GOLDEN Retriever generates a query q_k from the question and the available context, $(q, d_1, \ldots, d_{k-1})$. This formulation allows the model to generate queries based on information revealed in the supporting facts (see Figure 2, for example).

We note that GOLDEN Retriever is much more efficient, scalable, and interpretable at retrieving gold documents compared to its neural retrieval counterparts. This is because GOLDEN Retriever does not rely on a QA-specific IR engine tuned to a specific dataset, where adding new documents or question types into the index can be extremely inefficient. Further, GOLDEN Retriever generates queries in natural language, making it friendly to human interpretation and verification. One core challenge in GOLDEN Retriever, however, is to train query generation models in an efficient manner, because the search space for potential queries is enormous and off-the-shelf IR engines are not end-to-end differentiable. We outline our solution to this challenge in the following sections.

3.3 Query Generation

For each reasoning step, we need to generate the search query given the original question q and some context of documents we have already retrieved (initially empty). This query generation problem is conceptually similar to the QA task in that they both map a question and some context to a target, only instead of an answer, the target here is a search query that helps retrieve the desired supporting document for the next reasoning step. Therefore, we formulate the query generation process as a question answering task.

To reduce the potentially large space of possible queries, we favor a QA model that extracts text spans from the context over one that generates free-form text as search queries. We therefore employ DrQA's Document Reader model (Chen et al., 2017), which is a relatively light-weight recurrent neural network QA model that has demonstrated success in few-document QA. We adapt it to query generation as follows.

For each reasoning step $k = 1, \ldots, S$, given a question q and some *retrieval context* C_k which *ideally* contains the gold supporting documents $d_1, \ldots, d_{k-1}$, we aim to generate a search query q_k that helps us retrieve d_k for the next reasoning step. A Document Reader model is trained to select a span from C_k as the query

$$q_k = G_k(q, C_k),$$

where G_k is the query generator. This query is then used to search for supporting documents, which are concatenated with the current retrieval context to update it

$$C_{k+1} = C_k \bowtie \text{IR}_n(q_k)$$

where $\text{IR}_n(q_k)$ is the top n documents retrieved from the search engine using q_k, and $C_1 = q$.[6] At the end of the retrieval steps, we provide q as question along with C_S as context to the final few-

[5] For notational simplicity, d_k denotes the supporting document needed to complete the k-th step of reasoning. We also assume that the goal of each IR query is to retrieve one and only one gold supporting document in its top n results.

[6] In the query result, the title of each document is delimited with special tokens <t> and </t> before concatenation.

Question	Hop 1 Oracle	Hop 2 Oracle
What government position was held by the woman who portrayed Corliss Archer in the film Kiss and Tell?	Corliss Archer in the film Kiss and Tell	Shirley Temple
Scott Parkin has been a vocal critic of Exxonmobil and another corporation that has operations in how many countries?	Scott Parkin	Halliburton
Are Giuseppe Verdi and Ambroise Thomas both Opera composers?	Giuseppe Verdi	Ambroise Thomas

Table 1: Example oracle queries on the HOTPOTQA dev set.

document QA component detailed in Section 3.5 to obtain the final answer to the original question.

To train the query generators, we follow the steps above to construct the retrieval contexts, but during training time, when d_k is not part of the IR result, we replace the lowest ranking document with d_k before concatenating it with C_k to make sure the downstream models have access to necessary context.

3.4 Deriving Supervision Signal for Query Generation

When deriving supervision signal to train our query generators, the potential search space is enormous for each step of reasoning even if we constrain ourselves to predicting spans from the context. This is aggravated by multiple hops of reasoning required by the question. One solution to this issue is to train the query generators with reinforcement learning (RL) techniques (*e.g.*, REINFORCE (Sutton et al., 2000)), where (Nogueira and Cho, 2017) and (Buck et al., 2018) are examples of one-step query generation with RL. However, it is computationally inefficient, and has high variance especially for the second reasoning step and forward, because the context depends greatly on what queries have been chosen previously and their search results.

Instead, we propose to leverage the limited supervision we have about the gold supporting documents $d_1, \ldots, d_S$ to narrow down the search space. The key insight we base our approach on is that at any step of open-domain multi-hop reasoning, there is some *semantic overlap* between the retrieval context and the next document(s) we wish to retrieve. For instance, in our *Armada* example, when the retrieval context contains only the question, this overlap is the novel itself; after we have expanded the retrieval context, this overlap becomes the name of the author, *Ernest Cline*. Finding this semantic overlap between the retrieval context and the desired documents not only reveals the chain of reasoning naturally, but also allows us to use it as the search query for retrieval.

Because off-the-shelf IR systems generally optimize for shallow lexical similarity between query and candidate documents in favor of efficiency, a good proxy for this overlap is locating spans of text that have high lexical overlap with the intended supporting documents. To this end, we propose a simple yet effective solution, employing several heuristics to generate candidate queries: computing the longest common string/sequence between the current retrieval context and the title/text of the intended paragraph ignoring stop words, then taking the contiguous span of text that corresponds to this overlap in the retrieval context. This allows us to not only make use of entity names, but also textual descriptions that better lead to the gold entities. It is also more generally applicable than question decomposition approaches (Talmor and Berant, 2018; Min et al., 2019b), and does not require additional annotation for decomposition.

Applying various heuristics results in a handful of candidate queries for each document, and we use our IR engine (detailed next) to rank them based on recall of the intended supporting document to choose one as the final oracle query we train our query generators to predict. This allows us to train the query generators in a fully supervised manner efficiently. Some examples of oracle queries on the HOTPOTQA dev set can be found in Table 1. We refer the reader to Appendix B for more technical details about our heuristics and how the oracle queries are derived.

Oracle Query vs Single-hop Query We evaluate the oracle query against the single-hop query, i.e., querying with the original question, on the HOTPOTQA dev set. Specifically, we compare the recall of gold paragraphs, because the greater the recall, the fewer documents we need to pass into the expensive neural multi-hop QA component.

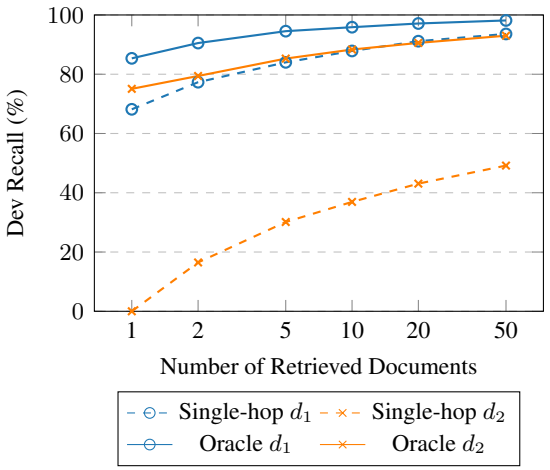

Figure 3: Recall comparison between single-hop queries and GOLDEN Retriever oracle queries for both supporting paragraphs on the HOTPOTQA dev set. Note that the oracle queries are much more effective than the original question (single-hop query) at retrieving target paragraphs in both hops.

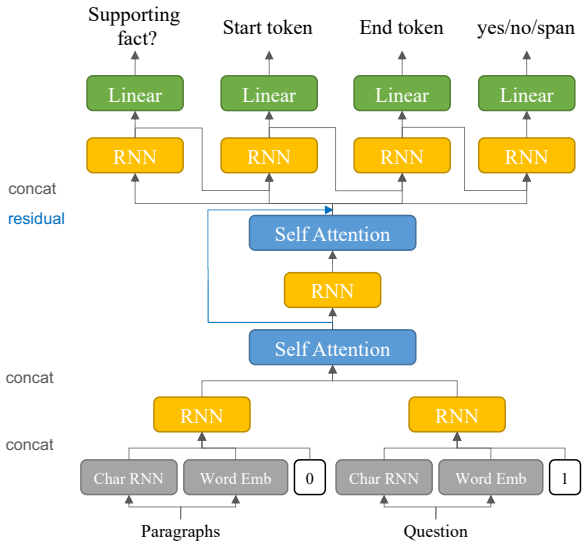

Figure 4: Question answering component in GOLDEN Retriever. (Best viewed in color)

We index the English Wikipedia dump with introductory paragraphs provided by the HOTPOTQA authors[7] with Elasticsearch 6.7 (Gormley and Tong, 2015), where we index the titles and document text in separate fields with bigram indexing enabled. This results in an index with 5,233,329 total documents. At retrieval time, we boost the scores of any search result whose title matches the search query better – this results in a better recall for entities with common names (*e.g.*, "*Armada*" the novel). For more details about how the IR engine is set up and the effect of score boosting, please refer to Appendix A.

In Figure 3, we compare the recall of the two gold paragraphs required for each question in HOTPOTQA at various number of documents retrieved (R@n) for the single-hop query and the multi-hop queries generated from the oracle. Note that the oracle queries are much more effective at retrieving the gold paragraphs than the original question in both hops. For instance, if we combine R@5 of both oracles (which effectively retrieves 10 documents from two queries) and compare that to R@10 for the single-hop query, the oracle queries improve recall for d_1 by 6.68%, and that for d_2 by a significant margin of 49.09%.[8] This means that the final QA model will need to consider far fewer documents to arrive at a decent set of supporting facts that lead to the answer.

3.5 Question Answering Component

The final QA component of GOLDEN Retriever is based on the baseline model presented in (Yang et al., 2018), which is in turn based on BiDAF++ (Clark and Gardner, 2017). We make two major changes to this model. Yang et al. (2018) concatenated all context paragraphs into one long string to predict span begin and end offsets for the answer, which is potentially sensitive to the order in which these paragraphs are presented to the model. We instead process them separately with shared encoder RNN parameters to obtain paragraph order-insensitive representations for each paragraph. Span offset scores are predicted from each paragraph independently before finally aggregated and normalized with a global softmax operation to produce probabilities over spans. The second change is that we replace all attention mechanism in the original model with self attention layers over the concatenated question and context. To differentiate context paragraph representations from question representations in this self-attention mechanism, we indicate question and context tokens by concatenating a 0/1 feature at the input layer. Figure 4 illustrates the QA model architecture.

[7] https://hotpotqa.github.io/wiki-readme.html

[8] Since HOTPOTQA does not provide the logical order its gold entities should be discovered, we simply call the document d_1 which is more easily retrievable with the queries, and the other as d_2.

System	Answer		Sup Fact		Joint	
	EM	F_1	EM	F_1	EM	F_1
Baseline (Yang et al., 2018)	25.23	34.40	5.07	40.69	2.63	17.85
GRN + BERT	29.87	39.14	13.16	49.67	8.26	25.84
MUPPET (Feldman and El-Yaniv, 2019)	30.61	40.26	16.65	47.33	10.85	27.01
CogQA (Ding et al., 2019)	37.12	48.87	22.82	57.69	12.42	34.92
PR-Bert	43.33	53.79	21.90	59.63	14.50	39.11
Entity-centric BERT Pipeline	41.82	53.09	26.26	57.29	17.01	39.18
BERT pip. (contemporaneous)	45.32	57.34	38.67	70.83	25.14	47.60
GOLDEN Retriever	37.92	48.58	30.69	64.24	18.04	39.13

Table 2: End-to-end QA performance of baselines and our GOLDEN Retriever model on the HOTPOTQA fullwiki test set. Among systems that were not published at the time of submission of this paper, "BERT pip." was submitted to the official HOTPOTQA leaderboard on May 15th (thus contemporaneous), while "Entity-centric BERT Pipeline" and "PR-Bert" were submitted after the paper submission deadline.

4 Experiments

4.1 Data and Setup

We evaluate our models in the fullwiki setting of HOTPOTQA (Yang et al., 2018). HOTPOTQA is a question answering dataset collected on the English Wikipedia, containing about 113k crowd-sourced questions that are constructed to require the introduction paragraphs of two Wikipedia articles to answer. Each question in the dataset comes with the two gold paragraphs, as well as a list of sentences in these paragraphs that crowdworkers identify as supporting facts necessary to answer the question. A diverse range of reasoning strategies are featured in HOTPOTQA, including questions involving missing entities in the question (our *Armada* example), intersection questions (*What satisfies property A and property B?*), and comparison questions, where two entities are compared by a common attribute, among others. In the few-document *distractor* setting, the QA models are given ten paragraphs in which the gold paragraphs are guaranteed to be found; in the open-domain *fullwiki* setting, which we focus on, the models are only given the question and the entire Wikipedia. Models are evaluated on their answer accuracy and explainability, where the former is measured as overlap between the predicted and gold answers with exact match (EM) and unigram F_1, and the latter concerns how well the predicted supporting fact sentences match human annotation (Supporting Fact EM/F_1). A joint metric is also reported on this dataset, which encourages systems to perform well on both tasks simultaneously.

We use the Stanford CoreNLP toolkit (Manning et al., 2014) to preprocess Wikipedia, as well as to generate POS/NER features for the query generators, following (Yang et al., 2018) and (Chen et al., 2017). We always detokenize a generated search query before sending it to Elasticsearch, which has its own preprocessing pipeline. Since all questions in HOTPOTQA require exactly two supporting documents, we fix the number of retrieval steps of GOLDEN Retriever to $S = 2$. To accommodate arbitrary steps of reasoning in GOLDEN Retriever, a stopping criterion is required to determine when to stop retrieving for more supporting documents and perform few-document question answering, which we leave to future work. During training and evaluation, we set the number of retrieved documents added to the retrieval context to 5 for each retrieval step, so that the total number of paragraphs our final QA model considers is 10, for a fair comparison to (Yang et al., 2018).

We include hyperparameters and training details in Appendix C for reproducibility.

4.2 End-to-end Question Answering

We compare the end-to-end performance of GOLDEN Retriever against several QA systems on the HOTPOTQA dataset: (1) the baseline presented in (Yang et al., 2018), (2) CogQA (Ding et al., 2019), the top-performing previously published system, and (3) other high-ranking systems on the leaderboard. As shown in Table 2, GOLDEN Retriever is much better at locating the correct supporting facts from Wikipedia compared

Setting	Ans F_1	Sup F_1	R@10*
GOLDEN Retriever	49.79	64.58	75.46
Single-hop query	38.19	54.82	62.38
HOTPOTQA IR	36.34	46.78	55.71

Table 3: Question answering and IR performance amongst different IR settings on the dev set. We observe that although improving the IR engine is helpful, most of the performance gain results from the iterative retrieve-and-read strategy of GOLDEN Retriever. (*: for GOLDEN Retriever, the 10 paragraphs are combined from both hops, 5 from each hop.)

System	Ans F_1	Sup F_1	Joint F_1
GOLDEN Retriever	49.79	64.58	40.21
w/ Hop 1 oracle	52.53	68.06	42.68
w/ Hop 1 & 2 oracles	62.32	77.00	52.18

Table 4: Pipeline ablative analysis of GOLDEN Retriever end-to-end QA performance by replacing each query generator with a query oracle.

to CogQA, as well as most of the top-ranking systems. However, the QA performance is handicapped because we do not make use of pretrained contextualization models (*e.g.*, BERT) that these systems use. We expect a boost in QA performance from adopting these more powerful question answering models, especially ones that are tailored to perform few-document multi-hop reasoning. We leave this to future work.

To understand the contribution of GOLDEN Retriever's iterative retrieval process compared to that of the IR engine, we compare the performance of GOLDEN Retriever against two baseline systems on the dev set: one that retrieves 10 supporting paragraphs from Elasticsearch with the original question, and one that uses the IR engine presented in HOTPOTQA.[9] In all cases, we use the QA component in GOLDEN Retriever for the final question answering step. As shown in Table 3, replacing the hand-engineered IR engine in (Yang et al., 2018) with Elasticsearch does result in some gain in recall of the gold documents, but that does not translate to a significant improvement in QA performance. Further inspection reveals that despite Elasticsearch improving overall recall of gold documents, it is only able to retrieve both gold documents for 36.91% of the dev set questions, in comparison to 28.21% from the IR engine in (Yang et al., 2018). In contrast, GOLDEN Retriever improves this percentage to 61.01%, almost doubling the recall over the single-hop baseline, providing the QA component a much better set of context documents to predict answers from.

Lastly, we perform an ablation study in which we replace our query generator models with our query oracles and observe the effect on end-to-end performance. As can be seen in Table 4, replacing G_1 with the oracle only slightly improves end-to-end performance, but further substituting G_2 with the oracle yields a significant improvement. This illustrates that the performance loss is largely attributed to G_2 rather than G_1, because G_2 solves a harder span selection problem from a longer retrieval context. In the next section, we examine the query generation models more closely by evaluating their performance without the QA component.

4.3 Analysis of Query Generation

To evaluate the query generators, we begin by determining how well they emulate the oracles. We evaluate them using Exact Match (EM) and F_1 on the span prediction task, as well as compare their queries' retrieval performance against the oracle queries. As can be seen in Table 6, the performance of G_2 is worse than that of G_1 in general, confirming our findings on the end-to-end pipeline. When we combine them into a pipeline, the generated queries perform only slightly better on d_1 when a total of 10 documents are retrieved (89.91% vs 87.85%), but are significantly more effective for d_2 (61.01% vs 36.91%). If we further zoom in on the retrieval performance on non-comparison questions for which finding the two entities involved is less trivial, we can see that the recall on d_2 improves from 27.88% to 53.23%, almost doubling the number of questions we have the complete gold context to answer. We note that the IR performance we report on the full pipeline is different to that when we evaluate the query generators separately. We attribute this difference to the fact that the generated queries sometimes retrieve both gold documents in one step.

To better understand model behavior, we also randomly sampled some examples from the dev set to compare the oracle queries and the predicted queries. Aside from exact matches, we find that the predicted queries are usually small variations of the oracle ones. In some cases, the model selects spans that are more natural and informative

[9] For the latter, we use the fullwiki test input file the authors released, which contains the top-10 IR output from that retrieval system with the question as the query.

	Question	Predicted q_1	Predicted q_2
(1)	What video game character did the voice actress in the animated film Alpha and Omega voice?	voice actress in the animated film Alpha and Omega *(animated film Alpha and Omega voice)*	Hayden Panettiere
(2)	What song was created by the group consisting of Jeffrey Jey, Maurizio Lobina and Gabry Ponte and released on 15 January 1999?	Jeffrey Jey *(group consisting of Jeffrey Jey, Maurizio Lobina and Gabry Ponte)*	Gabry Ponte and released on 15 January 1999 *("Blue (Da Ba Dee)")*
(3)	Yau Ma Tei North is a district of a city with how many citizens?	Yau Ma Tei North	Yau Tsim Mong District of Hong Kong *(Hong Kong)*
(4)	What company started the urban complex development that included the highrise building, The Harmon?	highrise building, The Harmon	CityCenter

Table 5: Examples of predicted queries from the query generators on the HOTPOTQA dev set. The oracle query is displayed in blue in parentheses if it differs from the predicted one.

Model	Span		R@5
	EM	F_1	
G_1	51.40	78.75	85.86
G_2	52.29	63.07	64.83

Table 6: Span prediction and IR performance of the query generator models for Hop 1 (G_1) and Hop 2 (G_2) evaluated separately on the HOTPOTQA dev set.

(Example (1) in Table 5). When they differ a bit more, the model is usually overly biased towards shorter entity spans and misses out on informative information (Example (2)). When there are multiple entities in the retrieval context, the model sometimes selects the wrong entity, which suggests that a more powerful query generator might be desirable (Example (3)). Despite these issues, we find that these natural language queries make the reasoning process more interpretable, and easier for a human to verify or intervene as needed.

Limitations Although we have demonstrated that generating search queries with span selection works in most cases, it also limits the kinds of queries we can generate, and in some cases leads to undesired behavior. One common issue is that the entity of interest has a name shared by too many Wikipedia pages (*e.g.*, "*House Rules*" the 2003 TV series). This sometimes results in the inclusion of extra terms in our oracle query to expand it (*e.g.*, Example (4) specifies that "*The Harmon*" is a highrise building). In other cases, our span oracle makes use of too much information from the gold entities (Example (2), where a human would likely query for "*Eiffel 65 song released 15 January 1999*" because "*Blue*" is not the only song mentioned in d_1). We argue, though, these are due to the simplifying choice of span selection for query generation and fixed number of query steps. We leave extensions to future work.

5 Conclusion

In this paper, we presented GOLDEN (Gold Entity) Retriever, an open-domain multi-hop question answering system for scalable multi-hop reasoning. Through iterative reasoning and retrieval, GOLDEN Retriever greatly improves the recall of gold supporting facts, thus providing the question answering model a much better set of context documents to produce an answer from, and demonstrates competitive performance to the state of the art. Generating natural languages queries for each step of reasoning, GOLDEN Retriever is also more interpretable to humans compared to previous neural retrieval approaches and affords better understanding and verification of model behavior.

Acknowledgements

The authors would like to thank Robin Jia among other members of the Stanford NLP Group, as well as the anonymous reviewers for discussions and comments on earlier versions of this paper. Peng Qi would also like to thank Suprita Shankar, Jamil Dhanani, and Suma Kasa for early experiments on BiDAF++ variants for HOTPOTQA. This research is funded in part by Samsung Electronics Co., Ltd. and in part by the SAIL-JD Research Initiative.

References

Christian Buck, Jannis Bulian, Massimiliano Ciaramita, Wojciech Gajewski, Andrea Gesmundo, Neil Houlsby, and Wei Wang. 2018. Ask the right questions: Active question reformulation with reinforce-

ment learning. In *Proceedings of the International Conference on Learning Representations*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*.

Jifan Chen and Greg Durrett. 2019. Understanding dataset design choices for multi-hop reasoning. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Christopher Clark and Matt Gardner. 2017. Simple and effective multi-paragraph reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association of Computational Linguistics*.

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

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2019. Question answering by reasoning across documents with graph convolutional networks. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

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 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. 2019. Cognitive graph for multi-hop reading comprehension at scale. In *Proceedings of the 57th Annual Meeting of the Association of Computational Linguistics*.

Yair Feldman and Ran El-Yaniv. 2019. Multi-hop paragraph retrieval for open-domain question answering. In *Proceedings of the 57th Annual Meeting of the Association of Computational Linguistics*.

Clinton Gormley and Zachary Tong. 2015. *Elasticsearch: The definitive guide: A distributed real-time search and analytics engine*. O'Reilly Media, Inc.

Ahmed Hassan Awadallah, Ryen W White, Patrick Pantel, Susan T Dumais, and Yi-Min Wang. 2014. Supporting complex search tasks. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 829–838. ACM.

Mandar Joshi, Eunsol Choi, Daniel S. 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*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Bernhard Kratzwald and Stefan Feuerriegel. 2018. Adaptive document retrieval for deep question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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

Rishabh Mehrotra, Prasanta Bhattacharya, and Emine Yilmaz. 2016. Deconstructing complex search tasks: A bayesian nonparametric approach for extracting sub-tasks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 599–605.

Rishabh Mehrotra and Emine Yilmaz. 2017. Extracting hierarchies of search tasks & subtasks via a bayesian nonparametric approach. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 285–294. ACM.

Sewon Min, Eric Wallace, Sameer Singh, Matt Gardner, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019a. Compositional questions do not necessitate multi-hop reasoning. In *Proceedings of the Annual Conference of the Association of Computational Linguistics*.

Sewon Min, Victor Zhong, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2019b. Multi-hop reading comprehension through question decomposition and rescoring. In *Proceedings of the Annual Conference of the Association of Computational Linguistics*.

Kyosuke Nishida, Itsumi Saito, Atsushi Otsuka, Hisako Asano, and Junji Tomita. 2018. Retrieve-and-read: Multi-task learning of information retrieval and reading comprehension. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 647–656. ACM.

Rodrigo Nogueira and Kyunghyun Cho. 2017. Task-oriented query reformulation with reinforcement learning.

Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document expansion by query prediction. *arXiv preprint arXiv:1904.08375*.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the 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 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at trec-3. *Nist Special Publication Sp*, 109:109.

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

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

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

Victor Zhong, Caiming Xiong, Nitish Shirish Keskar, and Richard Socher. 2019. Coarse-grain fine-grain coattention network for multi-evidence question answering. In *Proceedings of the International Conference on Learning Representations*.

A Elasticsearch Setup

A.1 Setting Up the Index

We start from the Wikipedia dump file containing the introductory paragraphs used in HOTPOTQA that Yang et al. (2018) provide,[10] and add the fields corresponding to Wikipedia page titles and the introductory paragraphs (text) into the index.

For the title, we use Elasticsearch's `simple` analyzer which performs basic tokenization and lowercasing of the content. For the text, we join all the sentences and use the `standard` analyzer which further allows for removal of punctuation and stop words. For both fields, we index an auxiliary field with bigrams using the `shingle` filter,[11] and perform basic `asciifolding` to map non ASCII characters to a similar ASCII character (*e.g.*, "é"→ "e").

At search time, we launch a `multi_match` query against all fields with the same query, which performs a full-text query employing the BM25 ranking function (Robertson et al., 1995) with all fields in the index, and returns the score of the best field for ranking by default. To promote documents whose title match the search query, we boost the search score of all title-related fields by 1.25 in this query.

A.2 Reranking Search Results

In Wikipedia, it is common that pages or entity names share rare words that are important to search engines, and a naive full-text search IR system will not be able to pick the one that matches the query the best. For instance, if one set up Elasticsearch according to the instructions above and searched for "*George W. Bush*", he/she would be surprised to see that the actual page is not even in the top-10 search results, which contains entities such as "*George W. Bush Childhood Home*" and "*Bibliography of George W. Bush*".

To this end, we propose to rerank these query results with a simple but effective heuristic that alleviates this issue. We would first retrieve at least 50 candidate documents for each query for consideration, and boost the query scores of documents whose title exactly matches the search query, or is a substring of the search query. Specifically, we multiply the document score by a heuristic constant between 1.05 and 1.5, depending on how well the document title matches the query, before reranking all search results. This results in a significant improvement in these cases. For the query "*George W. Bush*", the page for the former US president is ranked at the top after reranking. In Table 7, we also provide results from the single-hop query to show the improvement from title score boosting introduced from the previous section and reranking.

IR System	R@10 for d_1	R@10 for d_2
Final system	87.85	36.91
w/o Title Boosting	86.85	32.64
w/o Reranking	86.32	34.77
w/o Both	84.67	29.55

Table 7: IR performance (recall in percentages) of various Elasticsearch setups on the HOTPOTQA dev set using the original question.

B Oracle Query Generation

We mainly employ three heuristics to find the semantic overlap between the retrieval context and the desired documents: longest common subsequence (LCS), longest common substring (LCSubStr), and overlap merging which generalizes the two. Specifically, the overlap merging heuristic looks for contiguous spans in the retrieval context that have high rates of overlapping tokens with the desired document, determined by the total number of overlapping tokens divided by the total number of tokens considered in the span.

In all heuristics, we ignore stop words and lowercase the rest in computing the spans to capture more meaningful overlaps, and finally take the span in the retrieval context that all the overlapping words are contained in. For instance, if the retrieval context contains "*the* GOLDEN *Retriever model on* HOTPOTQA" and the desired document contains "GOLDEN *Retriever on the* HOTPOTQA *dataset*", we will identify the overlapping terms as "GOLDEN", "*Retriever*", and "HOTPOTQA", and return the span "GOLDEN *Retriever model on* HOTPOTQA" as the resulting candidate query.

To generate candidates for the oracle query, we apply the heuristics between combinations of {cleaned question, cleaned question without punctuation} × {cleaned document title, cleaned paragraph}, where cleaning means stop word removal and lowercasing. Once oracle queries are gener-

[10] https://hotpotqa.github.io/wiki-readme.html

[11] https://www.elastic.co/guide/en/elasticsearch/reference/6.7/analysis-shingle-tokenfilter.html

Hyperparameter	Values
Learning rate	$\underline{5 \times 10^{-4}}$, $\mathbf{1 \times 10^{-3}}$
Finetune embeddings	0, 200, **500**, *1000*
Epoch	**25**, *40*
Batch size	***32***, 64, 128
Hidden size	64, ***128***, 256, 512, 768
Max sequence length	15, *20*, 30, **50**, 100
Dropout rate	0.3, 0.35, ***0.4***, 0.45

Table 8: Hyperparameter settings for the query generators. The final hyperparameters for the Hop 1 query generator are shown in **bold**, and those for the Hop 2 query generator are shown in *underlined italic*.

ated, we launch these queries against Elasticsearch to determine the rank of the desired paragraph. If multiple candidate queries are able to place the desired paragraph in the top 5 results, we further rank the candidate queries by other metrics (*e.g.*, length of the query) to arrive at the final oracle query to train the query generators. We refer the reader to our released code for further details.

C Training Details

C.1 Query Generators

Once the oracle queries are generated, we train our query generators to emulate them on the training set, and choose the best model with F_1 in span selection on the dev set. We experiment with hyperparameters such as learning rate, training epochs, batch size, number of word embeddings to finetune, among others, and report the final hyperparameters for both query generators (G_1 and G_2) in Table 8.

C.2 Question Answering Model

Our final question answering component is trained with the paragraphs produced by the oracle queries (5 from each hop, 10 in total), with d_1 and d_2 inserted to replace the lowest ranking paragraph in each hop if they are not in the set already.

We develop our model based on the baseline model of Yang et al. (2018), and reuse the same default hyperparameters whenever possible. The main differences in the hyperparameters are: we optimize our QA model with Adam (with default hyperparameters) (Kingma and Ba, 2015) instead of stochastic gradient descent with a larger batch size of 64; we anneal the learning rate by 0.5 with a patience of 3 instead of 1, that is, we multiply the learning rate by 0.5 after three consecutive failures to improve dev F_1; we clip the gradient down to a maximum ℓ_2 norm of 5; we apply a 10% dropout to the model, for which we have increased the hidden size to 128; and use 10 as the coefficient by which we multiply the supporting facts loss, before mixing it with the span prediction loss. We configure the model to read 10 context paragraphs, and limit each paragraph to at most 400 tokens including the title.