# MULTI-STEP REASONING FOR OPEN-DOMAIN QUESTION ANSWERING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

This paper introduces a new framework for open-domain question answering in which the retriever and the reader *iteratively* interact with each other. The framework is agnostic to the architecture of the machine reading model and the retriever uses fast nearest neighbor search algorithms that allow it to scale to corpora containing millions of paragraphs. We show the efficacy of our architecture by achieving state-of-the-art results on large open domain datasets such as TriviaQA-unfiltered (Joshi et al., 2017). We also show that our multi-step-reasoning framework brings uniform improvements when applied to two different reader architectures.

## 1 INTRODUCTION

Open-domain question answering (QA) (Voorhees et al., 1999) involves a *retriever* for selecting relevant context from a large corpora of text (e.g. Wikipedia) and a *reader* for 'reasoning' on the retrieved context. A lot of effort has been put into designing sophisticated neural machine reading architectures for reading short context (e.g. a single paragraph), with much success (Wang & Jiang, 2017; Seo et al., 2017; Xiong et al., 2017; Wang et al., 2018c; Yu et al., 2018, inter alia). However, the performance of such systems degrade significantly when combined with a retriever in open domain settings (Chen et al., 2017). For example, the exact match accuracy of DrQA (Chen et al., 2017), on the SQUAD dataset (Rajpurkar et al., 2016) degrades from 69.5% to 28.4% in open-domain settings. The primary reason for this degradation in performance is due to the retriever's failure to find the relevant paragraphs for the machine reading model (Htut et al., 2018).

We propose the following desiderata for a general purpose open-domain QA system - (a) The retriever model should be *fast*, since it has to find the relevant context from a very large text corpora and give it to the more sophisticated and computationally expensive machine reading model (b) Secondly, the retriever and reader models should be interactive, i.e. if the reader model is unable to find the answer from the initial retrieved context, the retriever should be able to *learn* to provide more relevant context to the reader. Open-domain QA systems such as $R^3$ (Wang et al., 2018a) and DS-QA (Lin et al., 2018) have sophisticated retriever models where the reader and retriever are jointly trained. However, their retriever computes *question-dependent* paragraph representation which is then encoded by running an expensive recurrent neural network over the tokens in the paragraph. Since the retriever has to rank a lot of paragraphs, this design would not scale well to large corporas. One the other hand, the retriever model of QA systems such as DrQA (Chen et al., 2017), Clark & Gardner (2018) are based on a tf-idf retriever, but they lack trainable parameters and are consequently unable to recover from mistakes.

This paper introduces an open domain architecture in which the retriever and reader iteratively interact with each other. Our model first pre-computes and caches representation of context (paragraph). These representations are independent of the query unlike recent architectures (Seo et al., 2017; Xiong et al., 2017; Wang et al., 2018a; Lin et al., 2018) and hence can be computed and stored offline. Given an input question, the retriever performs *fast* inner product search to find the most relevant contexts. The highest ranked contexts are then passed to the neural machine reader which usually is a multi-layer recurrent neural network with attention. Our architecture is agnostic to the choice of the reader architecture and we show that doing multi-step-reasoning increases performance of two state-of-the-art machine reading architectures - DrQA (Chen et al., 2017) and BiDAF (Seo et al., 2017).

It is possible that the answer might not exist in the initial retrieved context or that the model would need to combine information across multiple contexts (Wang et al., 2018b). We equip the reader with

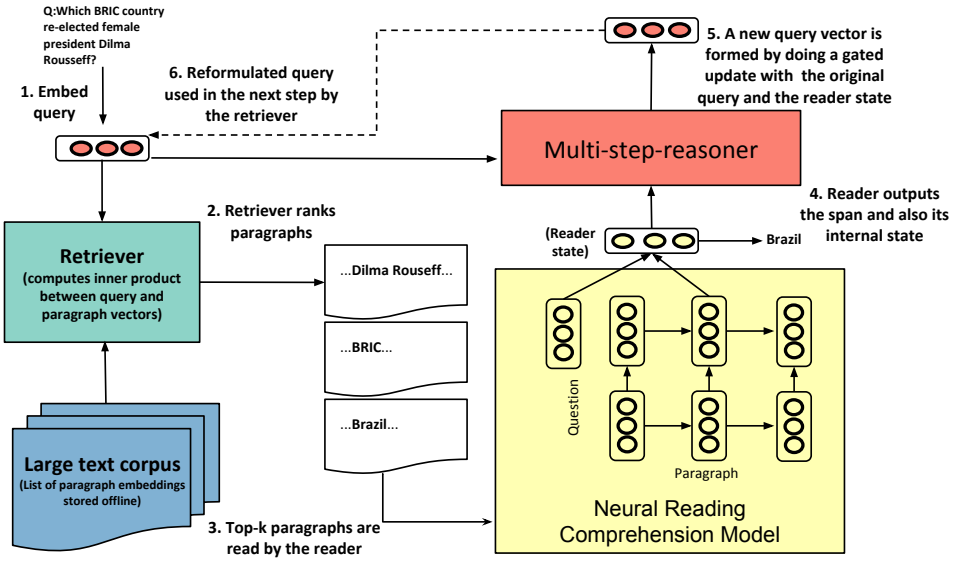


Figure 1: Overview of our open-domain architecture. The initial query is encoded and the retriever ranks *all* the paragraphs in the corpus. The $k$ top-ranked paragraphs are sent to the neural machine reader which outputs an answer span and also its internal reader state. The *multi-step-reasoner* component of our model takes in input the reader state and the previous query vector and does a gated update to produce a *reformulated* query. This new query vector is used by the retriever to re-rank the paragraphs in the corpus. Thus the multi-step-reasoner facilitates *iterative* interaction between the retriever (search engine) and the reader (QA model)

an additional gated recurrent unit (Cho et al., 2014; Chung et al., 2014) which takes in the state of the reader and the current query vector and *generates a new query vector*. This new query vector is then used by the retriever model to re-rank the context. This allows the model to read new contexts and combine evidence across multiple contexts, if required. Since the retriever makes a 'hard selection' of context to send to the reader, we train the retriever and the reader jointly using reinforcement learning (RL). Recently, Buck et al. (2018) introduced the ActiveQA model in which a RL agent sits between the user and the QA system and learns to reformulate the natural language query so as to maximize performance. However, reformulating the query in a natural language domain is a hard problem as we do not have a good generative model for natural language. In our approach, we circumvent this issue by performing *query reformulation* in the continuous embedding space and not in the space of natural language. Empirically, we find that our model significantly outperforms ActiveQA.

Our architecture draws inspiration from how students are instructed to take reading comprehension tests (Cunningham & Shablak, 1975; Bishop et al., 2006; Duggan & Payne, 2009). Given a document containing multiple paragraphs and a set of questions which can be answered from the document, (a) the student quickly skims the paragraphs, (b) then for each question, she finds the most relevant paragraphs that she thinks will answer the question. (c) She then carefully reads the chosen paragraph to answer the question (d) However, if the chosen paragraph does not answer the question, then given the question and the knowledge of what she has read till now, she decides which paragraph to read next. Step (a) is akin to our model encoding and storing the question independent paragraph representations and step (b) corresponds to the inner product search to find the relevant context. The reading of the context by the sophisticated neural machine reader corresponds to step (c) and the last step corresponds to the iterative (multi-step) interaction between the retriever and the reader.

To summarize, this paper makes the following contributions: (a) We introduce a new framework for open-domain QA in which the retriever and reader *iteratively* interact with each other via a novel multi-step-reasoning component which learns to reformulate the query vector. (b) Unlike several recent open-domain models, our paragraph vectors are independent of the query representations which makes our architecture highly scalable and we empirically demonstrate it by running large scale experiments over millions of paragraphs. (c) Lastly, our framework is agnostic to the architecture of

the reader and we demonstrate the efficacy of our framework on two different neural machine reading comprehension models.

## 2 MODEL

The architecture of our model consists of three main components - (a) paragraph retriever - that computes a relevance score for each paragraph w.r.t a given query and ranks them according to the computed score. (b) reader - a more sophisticated neural machine reading model that receives few top-ranked paragraphs from the retriever and outputs a span of text as a possible answer to the query and (c) multi-step-reasoner - a gated recurrent unit that facilitates iterative interaction between the retriever and the reader.

Formally, the input to our model is a natural language question $\mathrm{Q} = \mathrm{q}_1, \mathrm{q}_2, \ldots, \mathrm{q}_n$ consisting of $n$ tokens and a set of paragraphs $\mathrm{P} = \{\mathrm{p}_1, \mathrm{p}_2, \ldots \mathrm{p}_K\}$. Our model extracts a span of text $a$ as the answer to the question from the paragraphs in P. Note that the set of paragraphs in P can be the paragraphs in a set of documents retrieved by a search engine (as in TRIVIAQA web-open (Joshi et al., 2017)) or it could be *all* the paragraphs in a large text corpus such as Wikipedia. Next we describe each individual component of our model.

### 2.1 PARAGRAPH RETRIEVER

The paragraph retriever computes a score for how likely a paragraph is to contain an answer to a given question. The paragraph representations are computed independent of the query and once computed, they are not updated. This allows us to cache the representations and store them offline. The relevance score of a paragraph is computed as a inner product between the paragraph and the query vectors. The paragraph and query representations are computed as follows.

Given a paragraph $\mathrm{p} = \{\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_m\}$ consisting of $m$ tokens, a multi-layer recurrent neural network encodes each tokens in the paragraph.

$$\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\} = \mathrm{RNN}\left(\{\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_m\}\right)$$

where $\mathbf{p}_j \in \mathbb{R}^{2d}$ encodes useful contextual information around the $j$-th token. Specifically we choose to use a multi-layer bidirectional long-short term memory network (LSTM) (Hochreiter & Schmidhuber, 1997) and take $\mathbf{p}_j$ as the hidden units in the last layer of the RNN. We concatenate the representation computed by the forward and the backward LSTM. To compute a single paragraph vector $\mathbf{p} \in \mathbb{R}^{2d}$ from all the token representations, we combine them using weights $b_j \in \mathbb{R}$

$$b_j = \frac{\exp(\mathbf{w} \cdot \mathbf{p_j})}{\sum_{j'} \exp(\mathbf{w} \cdot \mathbf{p}_{j'})}$$
$$\mathbf{p} = W_s \sum_{j'} b_{j'} \cdot \mathbf{p_{j'}}$$

Here $b_j$ encodes the importance of each token and $\mathbf{w} \in \mathbb{R}^{2d}, \mathbf{W_s} \in \mathbb{R}^{2d \times 2d}$ are learned weights. This is similar to computing self-attention (Lin et al., 2017; Vaswani et al., 2017).

The query $\mathrm{q} = \{\mathrm{q}_1, \mathrm{q}_2, \ldots, \mathrm{q}_n\}$ is encoded by another network with the exact same architecture to obtain a query vector $\mathbf{q} \in \mathbb{R}^{2d}$. Next the relevance score of a paragraph w.r.t the query $(\mathrm{score}\,(\mathbf{p}, \mathbf{q}) \in \mathbb{R})$ is computed by a simple inner product:

$$\mathrm{score}\,(\mathbf{p}, \mathbf{q}) = \langle \mathbf{p}, \mathbf{q} \rangle \tag{1}$$

To return, we select $k$ paragraphs corresponding the maximum score. Thus, our problem reduces to maximum inner product search (MIPS). When number of candidate paragraphs is small, we can evaluate the score for each candidate paragraph on GPU to perform MIPS. However, such approach of linear search is wasteful and not scalable when the number of candidate paragraphs is large, e.g. in case of full Wikipedia. Borrowing ideas from Bachrach et al. (2014); Anonymous (2018), we propose to use excellent nearest neighbor (NN) search algorithms to perform the MIPS in sublinear time. Assume, we can have an upper bound $u$ for the L2 norm, i.e. $u \leq \|p\|, \forall p$. Then, the trick is to perform NN search in L2 distance with modified query and paragraph vectors so that the search is equivalent to MIPS for original query. In this regard, define the augmented paragraph vectors as

$\tilde{p}_i = [p_i; \sqrt{u^2 - \|p_i\|^2}]$ and augmented query vector as $\tilde{q} = [q; 0]$. With this transformation, we can see that:

$$\begin{aligned} \text{MIPS}(q|p) &= \arg\max_i \langle p_i, q \rangle = \arg\min_i u^2 + \|q\|^2 - 2\langle p_i, q \rangle \\ &= \arg\min_i \|\tilde{p}_i - \tilde{q}\| = \arg\min_i \|\tilde{x}_i - \tilde{q}\|^2 = \text{NN}(\tilde{q}|\tilde{p}) \end{aligned} \tag{2}$$

Many exact NN search algorithms can find k-NN in time (sublinear) logarithmic in number of paragraphs (Beygelzimer et al., 2006; Anonymous, 2018), after an one-time preprocessing. In case of question answering, the preprocessing can be done only once because the set of all paragraphs is fixed for all the queries. We chose to use SGTree (Anonymous, 2018) to perform the NN/MIPS due to its fast construction time and competitive performance.

Note that our paragraph representations are independent of the given query and are *never updated* once they are trained. After training is completed, we cache the $\mathbf{W_s p} \in \mathbb{R}^{2d}$ vectors. This is unlike many recent work in open-domain QA. For example, in $R^3$ (Wang et al., 2018a), the retriever uses Match-LSTM model (Wang & Jiang, 2017) to compute question-matching representations for each token in the passage. The paragraph representations are then obtained by running a bi-directional RNN over these matched representation. Although powerful, this architecture design will not scale in open-domain settings where the retriever has to re-compute new representations for possibly millions of paragraphs for every new question. In contrast, once training is over, we cache the paragraph representation and use it for every query at test time.

**Training** - Following previous work (Htut et al., 2018; Lin et al., 2018; Wang et al., 2018a), we gather labels for paragraphs during training using distant supervision (Mintz et al., 2009). A paragraph that contains the exact ground truth answer string is labeled as an positive example. For a positive (negative) labeled paragraph, we maximize (minimize) the $\log(\sigma(\text{score}(\text{p}, \text{q})))$. The number of layers of the bi-directional LSTM encoder is set to three and we use Adam (Kingma & Ba, 2014) for optimization. Once training is done, we pre-compute and cache the paragraph representation of each dataset in our experiments.

### 2.2 Machine Reader

The reader is a sophisticated neural machine reading comprehension (MRC) model that takes in the top few paragraphs sent by the retriever and outputs a span of answer text. Our model is agnostic to the exact architecture of the reader and we perform experiments to show the efficacy of our model on two state-of-the-art neural machine reading models - DrQA (Chen et al., 2017) and BiDAF (Seo et al., 2017).

MRC models designed for SQuAD (Rajpurkar et al., 2016), NewsQA (Trischler et al., 2016), etc operate on a single paragraph. However it has been shown that reading and aggregating evidence across multiple paragraphs is important when doing QA over larger evidence set (e.g. full Wikipedia document) (Swayamdipta et al., 2018; Clark & Gardner, 2018) and in open domain settings (Wang et al., 2018b). Most MRC models compute a start and end scores for each token in the paragraph that represents how likely a token is the start/end of an answer span. To gather evidence across multiple paragraphs sent by the retriever, we normalize the start/end scores across the paragraphs. Furthermore a text span can appear multiple times in a paragraph. To give importance to all answer spans in the text, our objective aggregates (sums) the log-probability of the score for each answer position. In other words, let $\mathcal{I}(w, p)$ denote the token start positions where the answer span appears in the paragraph $p$ and let $w_S$ be the starting word of the answer span. Our model maximizes the sum of the following objective for the start and end word of an answer spans as follows. (For brevity, we only show the objective for the starting word ($\text{w}_s$) of the span.)

$$\log\left(\frac{\sum_{j\in\mathcal{P}}\sum_{k\in\mathcal{I}(w_s,p_j)}\exp(\text{score}_{\text{start}}(k,j))}{\sum_{j\in\mathcal{P}}\sum_{i=1}^{n_j}\exp(\text{score}_{\text{start}}(i,j))}\right)$$

Here, $\mathcal{P}$ denotes the set of all top-ranked paragraphs by the retriever, $n_j$ denotes the number of tokens in paragraph $j$ and $\text{score}_{\text{start}}(k, j)$ denotes the start score of the $k$-th token in the $j$-th paragraph. A similar score aggregation strategy has been used in previous work (Kadlec et al., 2016; Clark & Gardner, 2018). During inference, following Chen et al. (2017); Seo et al. (2017), the score of a

span that starts at token position $i$ and ends at position $j$ of paragraph $p$ is given by the sum of the $\text{score}_{\text{start}}(i,p)$ and $\text{score}_{\text{end}}(j,p)$.

It should be noted that the changes we made to aggregate evidence over multiple paragraphs and mentions of spans needs *no change* in the original architecture of the machine reading model.

### 2.3 MULTI-STEP-REASONER

A novel component of our open-domain architecture is the multi-step-reasoner which facilitates *iterative interaction* between the retriever and the machine reader. The multi-step-reasoner is a gated recurrent unit (Cho et al., 2014; Chung et al., 2014) which takes in the current state of the reader and the current query vector and does a gated update to produce a reformulated query. The reformulated query is sent back to the retriever which uses it to re-rank the paragraphs in the corpus. Since the gated update of the multi-step-reasoner, conditions on the current state of the machine reader, this multi-step interaction provides a way for the search engine (retriever) and the QA model (reader) to communicate with each other. This can be seen as an instance of two agents cooperating via communication to solve a task (Lazaridou et al., 2017; Lee et al., 2018; Cao et al., 2018).

More formally, let $\mathbf{q_t} \in \mathbb{R}^{2d}$ be the current query representation which was most recently used by the paragraph *retriever* to score the paragraphs in the corpus (equation 1). The multi-step-reasoner also has access to the *reader* state which is computed from the hidden memory vectors of the reader. The reader state captures the current information that the reader has encoded after reading the paragraphs that was sent by the retriever. Next we show how the reader state is computed.

Let $\mathbf{m_j} \in \mathbb{R}^{2p}$ be the hidden vector associated with the $j$-th token in the paragraph as computed by the bi-directional multi-layer recurrent neural network encoder of the reader model[1]. Let $\mathbf{L} \in \mathbb{R}^{2p}$ be the final query representation of the reader model. $\mathbf{L}$ is usually created by some pooling operation on the hidden representation of each question token. The reader state $\mathbf{S} \in \mathbb{R}^{2p}$ is computed from each of the hidden vectors $\mathbf{m_j}$ and $\mathbf{L}$ by first computing soft-attention weights between each paragraph token, followed by combining each $\mathbf{m_j}$ with the soft attention weights.

$$\alpha_{\mathbf{j}} = \frac{\exp\left(\mathbf{m_j} \cdot \mathbf{L}\right)}{\sum_{j'} \exp\left(\mathbf{m'_j} \cdot \mathbf{L}\right)}$$

$$\mathbf{S} = \sum_j \left(\alpha_{\mathbf{j}} \cdot \mathbf{m_j}\right)$$

Finally, the new reformulated query $\mathbf{q_{t+1}}$ for the paragraph retriever is calculated by a GRU update.

$$\mathbf{q_{t+1}} = \text{GRU}\left(\mathbf{q_t}, \mathbf{S}\right)$$

The gated update ensures that relevant information from $\mathbf{q_t}$ is preserved and new and useful information from the reader state $\mathbf{S}$ is added to the reformulated query.

**Training** - There exists no supervision for training the query reformulation of the multi-step-reasoner. Therefore, to train the parameters of the GRU network we employ *reinforcement learning*. We define the problem as a deterministic finite horizon Partially Observed Markov decision process (POMDP). The components of POMDP are described as follows:

**States**. A state in the state space consists of the entire text corpora, the query, the answer, and $k$ selected paragraphs. The reader is part of the environment and is fully described given the current state, i.e. selected paragraphs and the query.

**Observations**. The agent only observes a function of the current state. In particular, to the agent only the initial query vector and the memory of the reader model is shown, which are a function of the current state. Intuitively, this represents the information encoded by the machine reader model after reading the top $k$ paragraphs sent by the retriever in current step.

[1] An overwhelming majority of successful MRC architectures use multi-layer RNN encoders. However there are exceptions such as QANET (Yu et al., 2018) and work by Swayamdipta et al. (2018). However these models still compute hidden memory vectors for each token position and a representation for the query and therefore the reader state can be computed in the same way.

**Algorithm 1** Multi-step reasoning for open-domain QA

**Input:** Question text $\text{Q}_{\text{text}}$, Text corpus (as a list of paragraphs) $\mathcal{T}$, Paragraph embeddings $\mathcal{P}$ (list of paragraph embeddings for each paragraph in $\mathcal{T}$), Model $\mathcal{M}$, Number of multi-steps T, number of top-ranked paragraphs $k$
**Output:** Answer span $a$

1: $\mathbf{q}_0 \leftarrow$ `encode_query`$(\text{Q}_{\text{text}})$ # (§ 2.1)
2: **for** t in `range`(T) **do**
3: $\{p_1, p_2, \ldots p_k\} \leftarrow \mathcal{M}$.`retriever.score_paras`$(\mathbf{q_t}, \mathcal{P}, k)$ # each $p_i$ denotes a paragraph id
4: $\text{P}_{\text{text}}^{1\ldots k} \leftarrow$ `get_text`$(\mathcal{T}, \{p_1, p_2, \ldots p_k\})$ # get text for the top paragraphs
5: $a_t$, $s_t$, $\mathbf{S_t} \leftarrow \mathcal{M}$.`reader.read`$(\text{P}_{\text{text}}^{1\ldots k}, \text{Q}_{\text{text}})$ # (§ 2.2); $\mathbf{S_t}$ denotes reader state
# $a_t$ denotes answer span
# $s_t$ denotes the score of the span
6: $\mathbf{q_t} \leftarrow \mathcal{M}$.multi_step_reasoner.GRU$(\mathbf{q_t}, \mathbf{S_t})$
# (§ 2.3); Query reformulation step
7: **end for**
8: **return** answer span $a$ with highest score

**Actions**. The set of *all* paragraphs in the text corpora forms the action space. The retriever scores all paragraphs w.r.t the current query and selects the top $k$ paragraphs to send to the reader model. We treat $k$ as a hyper-parameter in our experiments.

**Reward**. At every step, the reward is measured by how well the answer extracted by the reader model matches to the ground-truth answer. We use the $\text{F}_1$ score (calculated by word overlap between prediction and ground-truth) as the reward at each step.

**Transition**. The environment evolves deterministically after reading the paragraphs sent by the retriever.

Our policy is parameterized by the GRU of the multi-step-reasoner and we directly optimize the parameters to maximize the expected reward. We treat the reward at each step equally and do not apply any discounting. We employ the REINFORCE (Williams, 1992) algorithm to train the GRU network.

### 2.4 PUTTING IT ALL TOGETHER

Our open-domain architecture is summarized above in Algorithm 1. Given a large text corpora (as a list of paragraphs), the corresponding paragraph embeddings (which can be trained by the procedure in (§ 2.1)) and hyper-parameters $(\text{T}, k)$, our model $\mathcal{M}$ returns a text span $a$ as answer. The multi-step interaction between the retriever and reader can be best understood by the `for` loop in line 2 of algorithm 1. The initial query $\mathbf{q_0}$ is first used to rank *all* the paragraphs in the corpus (line 3), followed by which the top $k$ paragraphs are sent to the reader (line 4). The reader returns the answer span (with an associated score for the span) and also its internal state (§ 2.3) (line 5). The GRU network then takes in the current query and the reader state to produce the updated query which is then passed to the retriever (line 6). The retriever uses this updated query to again re-rank the paragraphs and the entire process is repeated for T steps. At the end of T steps, the model returns the span with the highest score returned by the reader model. The reader is trained using supervised learning (using the correct spans as supervision) and the parameters of the GRU network are trained using reinforcement learning. During training, we first pre-train the reader model by setting the number of multi-step reasoning steps ($\text{T} = 1$). After the training converges, we freeze the parameters of the reader model and train the parameters of the GRU network using policy gradients. The output of the reader model is used to generate the reward which is used to train the policy network.

## 3 RELATED WORK

Open domain QA is a well-established task that dates back to few decades of reasearch. For example the BASEBALL system (Green Jr et al., 1961) aimed at answering open domain question albeit for a specific domain. Open-domain QA has been popularized by the Trec-8 task (Voorhees et al.,

| Datasets | #q(test) | #p(test) | #p/q(test) |
|---|---|---|---|
| SearchQA | 27,247 | 1,351,493 | 49.6 |
| QuasarT | 3,000 | 299,389 | 99.8 |
| TriviaQA-unfiltered | 10,790 | 1,610,186 | 149 |
| TriviaQA-full | 11,274 | 1,684,193 | **1,684,193** |

Table 1: Statistics of various dataset. In TriviaQA full our retriever operates on 1.6M paragraphs

1999). Research has still progressed in open-domain settings with the introduction of datasets such as SearchQA Dunn et al. (2017), Quasar (Dhingra et al., 2017), TriviaQA (Joshi et al., 2017) but performance is still unsatisfactory. In many open-domain systems (Chen et al., 2017; Clark & Gardner, 2018), the retriever is a simple IR based system (e.g. tfidf retriever) with no trainable parameters and hence the retriever cannot overcome from its mistakes. Recent work such as $R^3$ (Wang et al., 2018a) use a trained retriever and have shown improvement in performance. However $R^3$ forms query dependent paragraph representation and such architectures will not scale to full open-domain settings where the retriever has to rank millions of paragraphs. The retriever and reader of $R^3$ and DS-QA Lin et al. (2018) are trained jointly but they do not support iterative reasoning thereby failing to recover from any mistakes made by the ranker or where evidence needs to be aggregated across multiple paragraphs. Models that do iterative reasoning (Shen et al., 2017; Liu et al., 2017) i.e. read a single paragraph multiple times, have proven to be effective in closed domain settings (e.g. SQuAD). Our model can be seen as doing iterative reasoning but in a more realistic, open-domain setting. In fact, our architecture reduces to these models when the set of evidence consists of only one paragraph and hence can be seen as a strict generalization of these models.

## 4 Experiments

We now present empirical studies for our model in order to establish that (i) multi-step reasoning is effective (§ 4.1), (ii) the framework achieves high accuracy on large open-domain datasets (§ 4.2), (iii) the framework is agnostic to reader architecture (§ 4.4), and (iv) the framework scales to corpora containing millions of paragraphs (§ 4.3) To illustrate these claims, we evaluate on the following open-domain question answering datasets:

**TriviaQA-open** – TriviaQA includes around 95,000 questions answers pairs authored by Trivia enthusiasts. We use the unfiltered TriviaQA corpus which contains a lot more paragraphs than the web/wiki setting. Moreover there is no guarantee that the evidence will have answer to a question.

**TriviaQA-full** – This is a setting in which we combine *all* evidence for every question in the development set. This resulted in a corpus containing **1.68M** paragraphs per question. In other words, to answer a question, the retriever has to find the correct context among 1.68M paragraphs.

**SearchQA** – is another open-domain dataset which consists of question-answer pairs crawled from the J! archive. The paragraphs are obtained from 50 web pages retrieved using the Google search API

**Quasar-T** – consists of 43K open-domain questions where the paragraphs are obtained by retrieving 50 sentences from the ClueWeb data source.

**Implementation** All the algorithms are implemented on PyTorch. For all experiments, we use Adam optimizer (Kingma & Ba, 2014) with the default hyperparameter settings. We run our experiments on a commodity machine with Intel® Xeon® CPU E5-2630 v4 CPU, 256GB RAM, and NVidia® Titan X GPU.

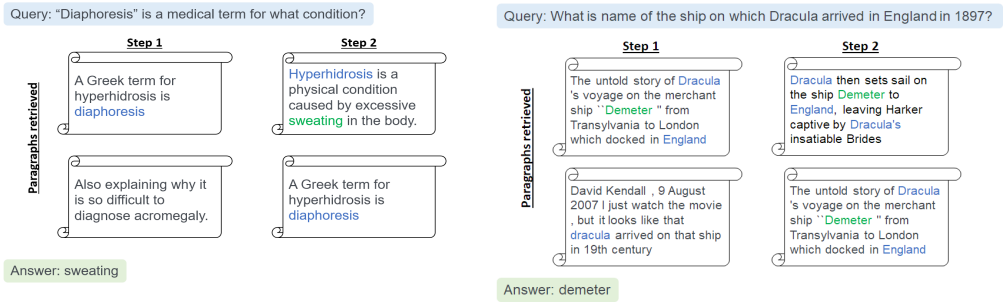


Figure 3: Examples of how our retriever iteratively modifies the query by reading context to fetch more relevant paragraphs

| Model | Quasar-T | | SearchQA | | TriviaQA | |
|---|---|---|---|---|---|---|
| | EM | F1 | EM | F1 | EM | F1 |
| GA (Dhingra et al., 2016) | 26.4 | 26.4 | - | - | - | - |
| Bidaf (Seo et al., 2017) | 25.9 | 28.5 | 28.6 | 34.6 | - | - |
| AQA (Buck et al., 2018) | - | - | 40.5 | 47.4 | - | - |
| $R^3$ (Wang et al., 2018a) | 35.3 | 41.7 | 49.0 | 55.3 | 47.3 | 53.7 |
| DS-QA [2] (Lin et al., 2018) | 37.27 | 43.63 | 58.8 | 64.5 | 48.7 | 56.3 |
| multi-step-reasoner | **39.53** | **46.67** | 55.01 | 61.61 | **51.94** | **61.66** |

[2] Despite our best effort we could not reproduce the results for Quasart-T using their code

Table 2: Performance on test sets for various datasets

### 4.1 Effectiveness of Multi-Step Reasoning

We show how multi-step reasoning using RL, as presented in Section 2.3, improves the question answering accuracy. For each dataset, we start with a reader model which has been trained using no multi-step reasoning steps. We freeze the parameters of the reader and then train the parameters of the GRU (policy) using reinforcement learning. Different datasets have varying degree of complexity of the questions, thus we need different number of multi-step reasoning. For every dataset, we uniformly see improvements in performance over the base model. The improvement with multi-step reasoning over the base model is shown in Figure 2 and illustratives are shown in Figure 3.

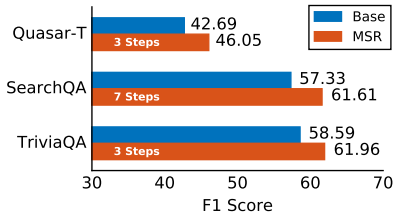


Figure 2: Performance gain (on development set) using multi step reasoning on 3 open domain datasets.

### 4.2 Accuracy

Next, we demonstrate high accuracy of multi-step-reasoner on reported baselines on the three challenging open datasets. In Table 2, we compare multi-step-reasoner with various prior art using Exact Match (EM) and F1 scores. On the hidden test set of TriviaQA we obtain the highest score ever reported to the best of our knowledge. We also have competitive scores on other datasets. For example on SearchQA, we beat AQA (Buck et al., 2018) significantly by more than 13 F1 points. This shows that it is much more efficient to do query reformulation in rich embedding space rather than in the space of natural language. We also consistently outperform $R^3$ on all datasets demonstrating the effectiveness of iterative reasoning.

| Models | F1 |
|---|---|
| BiDaF | 31.74 |
| BIDAF-multi-step-3 | 32.25 |
| BIDAF-multi-step-5 | 33.09 |

Table 3: Performance of BiDAF on the Quasar-T dataset.

### 4.3 SCALABILITY

Next we show the scalability of our architecture. For this experiment, we combine *all* paragraphs in the development set resulting in 1.68M paragraphs. Since we do not update paragraph representations and we use fast nearest neighbor data structures, our architectures can scale to such large corpus size. Since no other baselines scaled to this size, we compare our multi-step-reasoner model with a reader model with no multi-steps. The reader model got an exact match score of 37.45 and F1 score of 42.16, where as the model with reasoning steps 3 got a score of 39.76 (Em) and 44.30 (F1). This also shows that multi-step reasoning is effective even in the large scale setting.

### 4.4 EXPERIMENTS ON BIDAF READER

Finally, we show that our framework can be used in exactly the same way for a different reader architecture. We use an open source implementation of BiDAF (Seo et al., 2017) and modify it to return reader state (with the scores). Without any tuning, we observed that multi-step-reasoning gave a boost in peformance on the QuASAR-T dataset.

## 5 CONCLUSION

In this paper, we introduced a new framework for open-domain question answering in which the retriever and the reader *iteratively* interact with each other. The resulting framework improved accuracy in question answering task by 15% as compared to previous results establishing new state-of-the-art results on the TriviaQA-unfiltered dataset(Joshi et al., 2017). We also show that our method can scale to corpora containing millions of paragraphs. Lastly, we show that our framework brings uniform improvements to two neural machine reading architectures - DrQA (Chen et al., 2017) and BiDAF (Seo et al., 2017).

## REFERENCES


Anonymous. Terrapattern. In *To appear*, 2018.

Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender systems*, pp. 257–264. ACM, 2014.

Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pp. 97–104. ACM, 2006.

Penny A Bishop, Cynthia Reyes, and Susanna W Pflaum. Read smarter, not harder: Global reading comprehension strategies. *The Reading Teacher*, 2006.

Christian Buck, Jannis Bulian, Massimiliano Ciaramita, Andrea Gesmundo, Neil Houlsby, Wojciech Gajewski, and Wei Wang. Ask the right questions: Active question reformulation with reinforcement learning. In *ICLR*, 2018.

Kris Cao, Angeliki Lazaridou, Marc Lanctot, Joel Z Leibo, Karl Tuyls, and Stephen Clark. Emergent communication through negotiation. *arXiv preprint arXiv:1804.03980*, 2018.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *ACL*, 2017.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. In *ACL*, 2018.

Dick Cunningham and Scott L Shablak. Selective reading guide-o-rama: The content teacher's best friend. *Journal of Reading*, 1975.

Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. *arXiv preprint arXiv:1606.01549*, 2016.

Bhuwan Dhingra, Kathryn Mazaitis, and William W Cohen. Quasar: Datasets for question answering by search and reading. *arXiv preprint arXiv:1707.03904*, 2017.

Geoffrey B Duggan and Stephen J Payne. Text skimming: The process and effectiveness of foraging through text under time pressure. *Journal of Experimental Psychology: Applied*, 2009.

Matthew Dunn, Levent Sagun, Mike Higgins, V Ugur Guney, Volkan Cirik, and Kyunghyun Cho. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*, 2017.

Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*. ACM, 1961.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Phu Mon Htut, Samuel R Bowman, and Kyunghyun Cho. Training a ranking function for open-domain question answering. *arXiv preprint arXiv:1804.04264*, 2018.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL*, 2017.

Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Text understanding with the attention sum reader network. *arXiv preprint arXiv:1603.01547*, 2016.

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

Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. Multi-agent cooperation and the emergence of (natural) language. In *ICLR*, 2017.

Jason Lee, Kyunghyun Cho, Jason Weston, and Douwe Kiela. Emergent translation in multi-agent communication. In *ICLR*, 2018.

Yankai Lin, Haozhe Ji, Zhiyuan Liu, and Maosong Sun. Denoising distantly supervised open-domain question answering. In *ACL*, 2018.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. In *ICLR*, 2017.

Xiaodong Liu, Yelong Shen, Kevin Duh, and Jianfeng Gao. Stochastic answer networks for machine reading comprehension. *arXiv preprint arXiv:1712.03556*, 2017.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, 2009.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *EMNLP*, 2016.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *ICLR*, 2017.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. In *KDD*, 2017.

Swabha Swayamdipta, Ankur P Parikh, and Tom Kwiatkowski. Multi-mention learning for reading comprehension with neural cascades. In *ICLR*, 2018.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. Newsqa: A machine comprehension dataset. *arXiv preprint arXiv:1611.09830*, 2016.

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*, 2017.

Ellen M Voorhees, others, et al. The trec-8 question answering track report. In *Trec*, 1999.

Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. In *ICLR*, 2017.

Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerald Tesauro, Bowen Zhou, and Jing Jiang. R3: Reinforced ranker-reader for open-domain question answering. In *AAAI*, 2018a.

Shuohang Wang, Mo Yu, Jing Jiang, Wei Zhang, Xiaoxiao Guo, Shiyu Chang, Zhiguo Wang, Tim Klinger, Gerald Tesauro, and Murray Campbell. Evidence aggregation for answer re-ranking in open-domain question answering. In *ICLR*, 2018b.

Wei Wang, Ming Yan, and Chen Wu. Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering. In *ACL*, 2018c.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. In *ICLR*, 2017.

Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. In *ICLR*, 2018.