Real-Time Open-Domain Question Answering with Dense-Sparse Phrase Index

Minjoon Seo[1,5*]   **Jinhyuk Lee**[6*]   **Tom Kwiatkowski**[2],
Ankur P. Parikh[2]   **Ali Farhadi**[1,3,4]   **Hannaneh Hajishirzi**[1,3]
University of Washington[1]   Google[2]   Allen Institute for AI[3]   XNOR.AI[4]
NAVER[5]   Korea University[6]
{minjoon,ali,hannaneh}@cs.washington.edu
{tomkwiat,aparikh}@google.com   jinhyuk_lee@korea.ac.kr

Abstract

Existing open-domain question answering (QA) models are not suitable for real-time usage because they need to process several long documents on-demand for every input query, which is computationally prohibitive. In this paper, we introduce query-agnostic *indexable* representations of document phrases that can drastically speed up open-domain QA. In particular, our dense-sparse phrase encoding effectively captures syntactic, semantic, and lexical information of the phrases and eliminates the pipeline filtering of context documents. Leveraging strategies for optimizing training and inference time, our model can be trained and deployed even in a single 4-GPU server. Moreover, by representing phrases as pointers to their start and end tokens, our model indexes phrases in the entire English Wikipedia (up to 60 billion phrases) using under 2TB. Our experiments on SQuAD-Open show that our model is on par with or more accurate than previous models with 6000x reduced computational cost, which translates into at least 68x faster end-to-end inference benchmark on CPUs. Code and demo are available at `nlp.cs.washington.edu/denspi`

1 Introduction

Extractive open-domain question answering (QA) is usually referred to the task of answering an arbitrary factoid question (such as "Where was Barack Obama born?") from a general web text (such as Wikipedia). This is an extension of the reading comprehension task (Rajpurkar et al., 2016) of selecting an answer phrase to a question given an evidence document. To make a scalable open-domain QA system, One can leverage a search engine to filter the web-scale evidence to a few documents, in which the answer span can be extracted using a reading comprehension model (Chen et al.,

2017). However, the accuracy of the final QA system is bounded by the performance of the search engine due to the pipeline nature of the search process. What is more, running a neural reading comprehension model (Seo et al., 2017) on a few documents is still computationally costly, since it needs to process the evidence document for every new question at inference time. This often requires multi-GPU-seconds or tens to hundreds of CPU-seconds – BERT (Devlin et al., 2019) can process only a few thousand words per second on an Nvidia V100 GPU.

In this paper, we introduce Dense-Sparse Phrase Index (DENSPI), an *indexable* query-agnostic phrase representation model for real-time open-domain QA. The phrase representations are indexed offline using efficient training and memory-efficient strategies for storage. During inference time, the input question is mapped to the same representation space, and the phrase with maximum inner product search is retrieved.

Our phrase encoding model combines both dense and sparse vectors, eliminating the pipeline filtering of the context documents. Dense vectors are effective for encoding local syntactic and semantic cues leveraging recent advances in contextualized text encoding (Devlin et al., 2019), while sparse vectors are superior at encoding precise lexical information such as term frequencies (Cheng et al., 2016). Independent encoding of the document phrases and the question enables real-time inference; there is no need to re-encode documents for every question. Encoding phrases as a function of their start and end tokens facilitates indexable representations with under 2TB for up to 60 billion phrases in Wikipedia. Further, approximate nearest neighbor search on indexable representations allows fast and direct retrieval in a web-scale environment.

Experiments on SQuAD-Open (Chen et al.,
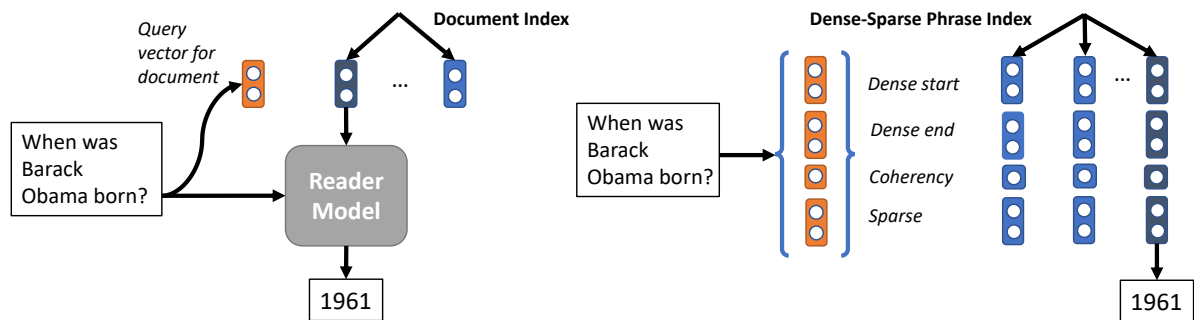
---

*Equal contribution.

Figure 1: An illustrative comparison between a pipelined QA system, e.g. DrQA (Chen et al., 2017) (left) and our proposed Dense-Sparse Phrase Index (right) for open-domain QA, best viewed in color. Dark blue vectors indicate the retrieved items from the index by the query.

2017) show that DENSPI is on par with or better than most state-of-the-art open-domain QA systems on Wikipedia with 6000x reduced computational cost on RAM. In our end-to-end benchmark, this translates into at least 68x faster query inference including disk access time.

At the web scale, every detail of the training, indexing, and inference needs to be carefully designed. For reproducibility under an academic setting, we discuss optimization strategies for reducing time and memory usage during each stage in Section 5. This enables us to start from scratch and fully deploy the model with a 4-GPU, 128GB memory, 2 TB PCIe[1] SSD server in a week.

2 Related Work

Open-domain question answering Creating a system that can answer an open-domain factoid question has been a significant interest to both academic and industrial communities. The problem is largely approached from two subfields: knowledge base (KB) and text (document) retrieval. Earlier work in large-scale question answering (Berant et al., 2013) has focused on answering questions from a structured KB such as Freebase (Bollacker et al., 2008). These approaches usually achieve a high precision, but their scope is limited to the ontology of the knowledge graph. While KB QA is undoubtedly an important part of open-domain QA, we mainly discuss literature in text-based QA, which is most relevant to our work.

Sentence-level QA has been studied since early 2000s, some of the most notable datasets being TrecQA (Voorhees and Tice, 2000) and WikiQA (Yang et al., 2015). See Prager et al. (2007)

for a comprehensive overview of early work. With the advancement of deep neural networks and the availability of massive QA datasets such as SQuAD (Rajpurkar et al., 2016), open-domain phrase-level question answering has gained a great popularity (Shen et al., 2017; Raiman and Miller, 2017; Min et al., 2018; Das et al., 2019), where a few (5-10) documents relevant to the question are retrieved and then a deep neural model finds the answer in the document. Most previous work on open-domain QA has focused on mitigating error propagation of retriever models in a pipelined setting (Chu-Carroll et al., 2012). For instance, retrieved documents could be re-ranked using reinforcement learning (Wang et al., 2018a), distant supervision (Lin et al., 2018), or multi-task learning (Nishida et al., 2018). Several studies have also shown that answer aggregation modules could improve performance of the pipelined models (Wang et al., 2018b; Lee et al., 2018).

Our work is motivated by Seo et al. (2018) and adopts the concept and the advantage of using phrase index for large-scale question answering, though they only experiment in a close-domain (vanilla SQuAD) setup.

Approximate similarity search Sublinear-time search for the nearest neighbor from a large collection of vectors is a significant interest to the information retrieval community (Deerwester et al., 1990; Blei et al., 2003). In metric space (L1 or L2), one of the most classic search algorithms is Locality-Sensitive Hashing (LSH) (Gionis et al., 1999), which uses a data-independent hashing function to map nearby vectors to the same cell. Stronger empirical performance has been observed with a data-dependent hashing function (Andoni and Razen-

---

[1]Disk random read access is a major bottleneck that PCIe over SATA is preferred.

shteyn, 2015) or k-means clustering for defining the cells. More recently, graph-based search algorithms (Malkov and Yashunin, 2018) have gained popularity as well. In non-metric space such as inner product, asymmetric Locality Sensitive Hashing (aLSH) (Shrivastava and Li, 2014) is considered, where maximizing inner product search can be transformed into minimizing L2 distance by appending a single dimension to the vectors. While these methods are widely used for dense vectors, for extremely sparse data (such as document tf-idf with stop words), it is often more efficient to construct an inverted index and only look up items that have common hot dimensions with the query.

Generative question answering Mapping the phrases in a document to a common vector space to that of the questions can be viewed as an exhaustive enumeration of all possible questions that can be asked on the document in the vector space, but without a surface-form decoder. It is worth noting that generative question answering (Lewis and Fan, 2019) has the opposite property; while it has a surface-form decoder by definition, it cannot easily enumerate a compact list of all possible semantically-unique questions.

Memory networks One can view the phrase index as a fixed external memory (Miller et al., 2016) where the key is the phrase vector and the value is the corresponding answer phrase span.

3 Overview

In this section, we formally define "open-domain question answering" and provide an overview of our proposed model.

3.1 Problem Definition

In this paper, we are interested in the task of answering factoid questions from a large collection of web documents in real-time. This is often referred to as open-domain question answering (QA). We formally formulate the task as follows. We are given a fixed set of (Wikipedia) documents $x^1, \ldots, x^K$ (where K is the number of documents, often on the order of millions), and each document x^k has N_k words, $x^k_1, \ldots, x^k_{N_k}$. The task is to find the answer a to the question $q = q_1, \ldots, q_S$. Then an open-domain QA model is a scoring function F for each candidate phrase span $x^k_{i:j}$ such that $a = \mathrm{argmax}_{k,i,j} F(x^k_{i:j}, q)$.

文档 x^k 的一个 span

Scalability challenge While the formulation is straightforward, argmax-ing over the entire corpus is computationally prohibitive, especially if F is a complex neural model. To avoid the computational bottleneck, previous open-domain QA models adopt pipeline-based methods; that is, as illustrated in Figure 1 left, a fast retrieval-based model is used (e.g. tf-idf) to obtain a few relevant documents to the question, and then a neural QA model is used to extract the exact answer from the documents (Chen et al., 2017). However, the method is not *efficient* enough for real-time usage because the neural QA needs to re-encode all the documents for every new question, which is computationally expensive even with modern GPUs, and not suitable for low-latency applications.

open-domain QA 的一般做法.

3.2 Encoding and Indexing Phrases

Motivated by Seo et al. (2018), our model encodes query-agnostic representations of text spans in Wikipedia offline and obtains the answer in real-time by performing nearest neighbor search at inference time. We represent each phrase span in the corpus (Wikipedia) with a dense vector and a sparse vector. The dense vector is effective for encoding syntactic and semantic cues, while the sparse vector is good at encoding precise lexical information. That is, the embedding of each span (i, j) in the document x^k is represented with

① 表示文档

对 x^k 文档的 span (i,j):

$$\mathbf{x}^k_{i:j} = [\mathbf{d}^k_{i:j}, \mathbf{s}^k_{i:j}] \in \mathbb{R}^{d^d + d^s} \qquad (1)$$

dense vector   sparse vector $d^d \leq d^s$

where $\mathbf{d}^k_{i:j} \in \mathbb{R}^{d^d}$ is the dense vector and $\mathbf{s}^k_{i:j} \in \mathbb{R}^{d^s}$ is the sparse vector for span (i, j) in the k-th document. Note that $d^d \ll d^s$. This is also illustrated in Figure 1 right. Text span embeddings $(\mathbf{x}^k_{i:j})$ for all possible i, j, k pairs with $j - i < J$, where J is maximum span length (i.e. all possible spans from all documents in Wikipedia), are precomputed and stored as a *phrase index*. Then at inference time, we embed each question into the same vector space, $\mathbf{q} = [\mathbf{d}', \mathbf{s}'] \in \mathbb{R}^{d^d + d^s}$. Finally, the answer to the question is obtained by finding the maximum inner product between $\mathbf{q}$ and $\mathbf{x}^k_{i:j}$,

所有符合条件的 $x^k_{i:j}$，提前计算所有的向量表达，并建索引.

② 表示 query

③ 计算内积

$$k^*, i^*, j^* = \underset{k,i,j}{\mathrm{argmax}} \; \mathbf{q} \cdot \mathbf{x}^k_{i:j}. \qquad (2)$$

Needlessly to say, designing a good phrase representation model is crucial, which will be discussed in Section 4. Also, while inner product search is much more efficient than re-encoding documents,

the search space is still quite large, such that exact search on the entire corpus is still undesirable. We discuss how we perform inner product search efficiently in Section 5.

4 Phrase and Question Embedding

In this section, we first explain the embedding model for the dense vector in Section 4.1. Then we describe the embedding model for the sparse vector in Section 4.2. Lastly, we describe the corresponding question embedding model to be queried on the phrase index in Section 4.3. For the brevity of the notations, we omit the superscript k in this section since we do not learn cross-document relationships.

4.1 Dense Model

The dense vector is responsible for encoding syntactic or semantic information of the phrase with respect to its context. We decompose the dense vector $\mathbf{d}_{i:j}^k$ (Equation 1) into three components: a vector that corresponds to the start position of the phrase, a vector that corresponds to the end position, and a scalar value that measures the coherency between the start and the end vectors. Representing phrases as a function of start and end vectors allows us to efficiently compute and store the vectors instead of enumerating all possible phrases (discussed in Section 5.2).[2]

The coherency scalar allows us to avoid non-constituent phrases during inference. For instance, consider a sentence such as "Barack Obama was the 44th President of the US. He was also a lawyer." and when a question "What was Barack Obama's job?" is asked. Since both answers "44th President of the US" and "lawyer" are technically correct, we might end up with the answer that spans from "44th" to "lawyer" if we model start and end vectors independently. The coherency scalar helps us avoid this by modeling it as a function of the start position and the end position. Formally, after phrase vector decomposition into dense and sparse, we can expand the dense vector into

$$\mathbf{d}_{i:j} = [\mathbf{a}_i, \mathbf{b}_j, c_{i,j}] \in \mathbb{R}^{2d^b+1} \qquad (3)$$

where $\mathbf{a}_i, \mathbf{b}_j \in \mathbb{R}^{d^b}$ are the start and end vectors for the i-th and j-th words of the document, respectively; and $c_{i,j} \in \mathbb{R}$ is the phrasal coherency

---

[2]Our phrase encoding is analogous to how existing QA systems obtain the answer by predicting its start and the end positions.

scalar between i-th and j-th positions (hence $d^d = 2d^b + 1$).

To obtain these components of the dense vector, we leverage available contextualized word representations, in particular BERT-large (Devlin et al., 2019), which is pretrained on a large corpus (Wikipedia and BookCorpus) and has proved to be very powerful in numerous natural language tasks. BERT maps a sequence of the document tokens $\boldsymbol{x} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ to a sequence of corresponding vectors (i.e. a matrix) $\mathbf{H} = [\mathbf{h}_1; \ldots; \mathbf{h}_N] \in \mathbb{R}^{N \times d}$, where N is the length of the input sequence, d is the hidden state size, and $[;]$ is vertical concatenation. We obtain the three components of the dense vector from these contextualized word representations.

We fine-tune BERT to learn a d-dimensional vector $\mathbf{h}_i$ for encoding each token $\boldsymbol{x}_i$. Every token encoding is split into four vectors $\mathbf{h}_i = [\mathbf{h}_i^1, \mathbf{h}_i^2, \mathbf{h}_i^3, \mathbf{h}_i^4] \in \mathbb{R}^d$, where $[,]$ is a column-wise concatenation. Then we obtain the dense start vector $\mathbf{a}_i$ from $\mathbf{h}_i^1$ and dense end vector $\mathbf{b}_j$ from $\mathbf{h}_j^2$. Lastly, we obtain the coherency scalar $c_{i,j}^k$ from the inner product of $\mathbf{h}_i^3$ and $\mathbf{h}_j^4$. The inner product allows more coherent phrases to have more similar start and end encodings. That is,

$$\mathbf{d}_{i:j} = [\mathbf{h}_i^1, \mathbf{h}_j^2, \mathbf{h}_i^3 \cdot \mathbf{h}_j^4] \in \mathbb{R}^{2d^b+1} \qquad (4)$$

where $\cdot$ indicates inner product operation and $\mathbf{h}_i^1, \mathbf{h}_j^2 \in \mathbb{R}^{d^b}$ and $\mathbf{h}_i^3, \mathbf{h}_j^4 \in \mathbb{R}^{d^c}$ (hence $2d^b + 2d^c = d$).

4.2 Sparse Model

We use term-frequency-based encoding to obtain the sparse embedding $\mathbf{s}_{i:j}^k$ for each phrase. Specifically, we largely follow DrQA (Chen et al., 2017) to construct 2-gram-based tf-idf, resulting in a highly sparse representation ($d^d \approx 16M$) for each document. The sparse vectors are normalized so that the inner product effectively becomes cosine similarity. We also compute a paragraph-level sparse vector in a similar way and add it to each document sparse vector for a higher sensitivity to local information. Note that, however, unlike DrQA where the sparse vector is merely used to retrieve a few (5-10) documents, we concatenate the sparse vector to the dense vector to form a standalone single phrase vector as in Equation 1.

4.3 Question Embedding Model

At inference, the question is encoded as $\mathbf{q} = [\mathbf{d}', \mathbf{s}'] = [\mathbf{a}', \mathbf{b}', c', \mathbf{s}']$ with the same number of

4433

components as the phrase index. To obtain the dense query vector $\mathbf{d}' = [\mathbf{a}', \mathbf{b}', c']$, we use a special token ([CLS] for BERT) which is appended to the front of the question words (i.e. input question words are $\boldsymbol{q} = [\text{CLS}], \boldsymbol{q}_1, \ldots, \boldsymbol{q}_S$). This allows us to model the dense query embedding differently from the dense embedding in the phrase index while sharing all parameters of the BERT encoder. That is, given the contextualized word representations of the question, we obtain the the dense query vector by

$$\mathbf{p}' = [\mathbf{h}_1'^1, \mathbf{h}_1'^2, \mathbf{h}_1'^3 \cdot \mathbf{h}_1'^4], \tag{5}$$

where $\mathbf{h}_1'^1$ is the encoding corresponding to the (first) special token and we obtain the others in a similar way. To obtain the sparse query vector $\mathbf{s}'$, we use the same tf-idf embedding model (Section 4.2) on the entire query.

5 Training, Indexing & Search

Open-domain QA is a web-scale experiment, dealing with billions of words in Wikipedia while aiming for real-time inference. Hence (1) training the models, (2) indexing the embeddings, and (3) performing inner product search at inference time are non-trivial for both (a) computational time and (b) memory efficiency. In particular, we carry out this section assuming that we have a constrained hardware environment of 4 P40 GPUs, 128 GB RAM, 16 cores and 2 TB of PCIe SSD storage, to promote reproducibility of our experiments under academic setting.[3]

5.1 Training

As discussed in Section 4.2, the sparse embedding model is trained in an unsupervised manner. For training the dense embedding model, instead of directly optimizing for Equation 2 on entire Wikipedia, which is computationally prohibitive, we provide the golden paragraph to each question during training (i.e. SQuAD v1.1 setting).

Given the dense phrase and question embeddings, we first expand Equation 2 by substituting Equation 4 and Equation 5 (omitting document

terms):

$$
\begin{aligned}
i^*, j^* &= \underset{i,j}{\operatorname{argmax}} \, \mathbf{d}' \cdot \mathbf{d}_{i:j} \\
&= \underset{i,j}{\operatorname{argmax}} \, \mathbf{h}_1'^1 \cdot \mathbf{h}_i^1 + \mathbf{h}_1'^2 \cdot \mathbf{h}_j^2 + \\
&\quad \mathbf{h}_1'^3 \cdot \mathbf{h}_1'^4 + \mathbf{h}_i^3 \cdot \mathbf{h}_j^4
\end{aligned}
$$

From now on we let $l_i^1 = \mathbf{h}_1'^1 \cdot \mathbf{h}_i^1$ (phrase start logits), $l_j^2 = \mathbf{h}_1'^2 \cdot \mathbf{h}_j^2$ (phrase end logits), and $l_{i,j} = l_i^1 + l_j^2 + \mathbf{h}_1'^3 \cdot \mathbf{h}_1'^4 + \mathbf{h}_i^3 \cdot \mathbf{h}_j^4$ i.e. the value that is being maximized in the above equation.

One straightforward way to define the loss is to define it as the negative log probability of the correct answer where $\Pr(i,j) \propto \exp(l_{i,j})$. In other words,

$$L = -l_{i^*,j^*} + \log \sum_{i,j} \exp(l_{i,j}) \tag{6}$$

where L is the loss to minimize. Note that explicitly enumerating all possible phrases (enumerating all (i,j) pairs) during training time would be memory-intensive. Instead, we can efficiently obtain the loss by:

$$
\begin{aligned}
\mathbf{l}^1 &= [l_1^1, \ldots, l_T^1] = \mathbf{q}^1 \mathbf{H}^{1\top} \\
\mathbf{l}^2 &= \mathbf{h}_1^2 \mathbf{H}^{2\top} \\
\mathbf{L} &= \mathbf{H}^3 \mathbf{H}^{4\top} + \mathbf{l}^{1\top} + \mathbf{l}^2
\end{aligned}
$$

where $\mathbf{H}^m = [\mathbf{h}_1^m, \ldots, \mathbf{h}_T^m]$ for $m = 1, 2, 3, 4, +$ is with broadcasting and (i,j)-th element of $\mathbf{L}$ is $l_{i,j}$. Note that L can be entirely computed from $\mathbf{L}$.

While the loss function is clearly unbiased with respect to $\Pr(i,j) \propto \exp(l_{i,j})$, the summation in Equation 6 is computed over T^2 terms which is quite large and causes small gradient. To aid training, we define an auxilary loss L^1 corresponding to the start logits,

$$L^1 = -l_{i^*}^1 + \log \sum_i \exp\left(\frac{1}{T} \sum_j l_{i,j}\right) \tag{7}$$

and L^2 for the end logits in a similar way. By early summation (taking the mean), we reduce the number of exponential terms and allow larger gradients. We average between the true and aux loss for the final loss: $\frac{L}{2} + \frac{L^1 + L^2}{4}$.

No-Answer Bias During training SQuAD (v1.1), we never observe negative examples (i.e. an unanswerable question in the paragraph). Following Levy et al. (2017), we introduce a trainable

---

[3]Training takes 16 hours (64-GPU hours) and indexing takes 5 days (500 GPU-hours).

no-answer bias when computing softmax. For each paragraph, we create two negative examples by bringing one question from another article and one question from the same article but different paragraphs. Instead of randomly sampling, we bring the question with the highest inner product (i.e. most similar) with a randomly-picked positive question in the current paragraph, using a question embedding model trained on SQuAD v1.1. We jointly train the positive examples with the negative examples.

5.2 Indexing

Wikipedia consists of approximately 3 billion tokens, so enumerating all phrases with length ≤ 20 will result in about 60 billion phrases. With 961D of `float32` per phrase, one needs 240 TB of storage (60 billion times 961 dimensions times 4 bytes per dimension). While not impossible in industry scale, the size is clearly out of reach for independent or academic researchers and critically unfriendly for open research. We discuss three techniques we employee to reduce the size of the index to 1.2 TB without sacrificing much accuracy, which becomes much more manageable for everyone. In practice, additional 300-500GB will be needed to store auxiliary information for efficient indexing, which still sums up to less than 2TB.

1. Pointer Since each phrase vector is the concatenation of $\mathbf{a}_i$ and $\mathbf{b}_j$ (and a scalar $c_{i,j}$ but it takes very little space), many phrases share the same start or end vectors. Hence we store a single list of the start and the end vectors independently and just store pointers to those vectors for the phrase representation. This effectively reduces the memory footprint from 240 TB to 12 TB.

2. Filtering We train a simple single-layer binary classifier on top of each of the start and end vectors, supervised with the actual answer (without observing the question). This allows us to not store vectors that are unlikely to be a potential start or end position of the answer phrase, further reducing the memory footprint from 12 TB to 5 TB.

3. Quantization We reduce the size of each vector by quantization. That is, we convert each `float32` value to `int8` with appropriate offset and scaling. This allows us to reduce the size by one-fourth. Hence the final memory consumption is 1.2 TB. In future, more advanced methods such as Product Quantization (Jegou et al., 2011) can be considered.

5.3 Search

While it would be ideal to (and possible to) directly approximate argmax in Equation 2 by using sparse maximum inner product search algorithm (some discussed in Section 2), we could not find a good open-source implementation that can scale up to billions of vectors and handle the dense and the sparse part of the phrase vector at the same time. We instead approximate the argmax by doing search on the dense vectors first and then reranking by accessing the corresponding sparse vectors.[4] For this, we use Faiss (Johnson et al., 2017), open-sourced and large-scale-friendly similarity search package for dense vectors.

Also, instead of directly searching on the dense vector $\mathbf{d}_{i:j}$ (concatenation of start, end, and coherency), we first search on the start vector $\mathbf{a}_i$ and obtain the best end position for each retrieved start position by computing the rest. We found that this allows us to save memory and time without sacrificing much accuracy, since the start vectors alone seem to contain sufficiently rich syntactic and semantic information already that makes the search possible even in a large scale.

6 Experiments

Experiment section is divided into two parts. First, we report results on SQuAD (Rajpurkar et al., 2016). This can be considered as a small-scale prerequisite to the open-domain experiment. It also allows a convenient comparison to state-of-the-art models in SQuAD, especially on the speed of the model. Under a fully controlled environment and batch-query scenario, our model processes words nearly 6000 times faster than DrQA (Chen et al., 2017). Second, we report results on Open-domain SQuAD (called SQuAD-Open), following the same setup as in DrQA. We show that our model achieves 3.6% better accuracy and nearly 68 times faster end-to-end inference time than previous work while exploring 100 times more unique documents. All experiments are CPU-only benchmark.

---

[4]Note that searching on dense first rather than sparse first (which is typical in many open-domain QA systems, where they retrieve top-k documents) implies a widely different behavior, as described in Section 6.2.

	Model	EM	F1	W/s
Original	DrQA	69.5	78.8	4.8K
	BERT-Large	84.1	90.9	51
Query-Agnostic	LSTM+SA	49.0	59.8	-
	LSTM+SA+ELMo	52.7	62.7	-
	DENSPI (dense only)	73.6	81.7	28.7M
	+ Linear layer	66.9	76.4	-
	+ Indep. encoders	65.4	75.1	-
	− Coherency scalar	71.5	81.5	-

Table 1: Results on SQuAD v1.1. 'W/s' indicates number of words the model can process (read) per second on a CPU in a batch mode (multiple queries at a time). DrQA (Chen et al., 2017) and BERT (Devlin et al., 2019) are from SQuAD leaderboard, and LSTM+SA and LSTM+SA+ELMo are query-agnostic baselines from Seo et al. (2018).

6.1 SQuAD v1.1 Experiments

In the SQuAD v1.1 setup, our model effectively uses only the dense vector since every sparse (document tf-idf) vector will be identical in the same paragraph. While this is a much easier problem than open-domain, it can serve as a reliable and fast indicator of how well the model would do in the open-domain setup.

Model details We use BERT-large ($d = 1024$) for the text encoders, which is pretrained on a large text corpus (Wikipedia dump and Book Corpus). We refer readers to the original paper by Devlin et al. (2019) for details; we mostly use the default settings described there. We use $d^b = 480$, resulting in phrase size of $2d^b + 1 = 961$, and $d^c = 32$. We train with a batch size of 12 (on four P40 GPUs) for 3 epochs.

Baselines We compare the performance of our system DENSPI with a few baselines in terms of accuracy and efficiency. The first group are among the models that are submitted to SQuAD v1.1 Leaderboard, specifically DrQA (Chen et al., 2017) and BERT (Devlin et al., 2019) (current state of the art). These models encode the evidence document given the question, but they suffer from the disadvantage that the evidence document needs to be re-encoded for every new question at the inference time, and they are strictly linear time in that they cannot utilize approximate search algorithms. The second group of baselines are introduced by Seo et al. (2018), specifically LSTM+SA and LSTM+SA+ELMo that also encode phrases independent of the question using LSTM, Self-

Attention, and ELMo (Peters et al., 2018) encodings.

Results Table 1 compares the performance of our system with different baselines in terms of efficiency and accuracy. We note the following observations from the result table. (1) DENSPI outperforms the query-agnostic baseline (Seo et al., 2018) by a large margin, 20.1% EM and 18.5% F1. This is largely credited towards the usage of BERT encoder with an effective phrase embedding mechanism on the top. (2) DENSPI outperforms DrQA by 3.3% EM. This signifies that phrase-indexed models can now outperform early (unconstrained) state-of-the-art models in SQuAD. (3) DENSPI is 9.2% below the current state of the art. The difference, which we call *decomposability gap*[5], is now within 10% and future work will involve further closing the gap. (4) Query-agnostic models can process (read) words much faster than query-dependent representation models. In a controlled environment where all information is in memory and the documents are pre-indexed, DENSPI can process 28.7 million words per second, which is 6,000 times faster than DrQA and 563,000 times faster than BERT without any approximation.

Ablations Ablations are also shown at the bottom of Table 1. The first ablation adds a linear layer on top of the BERT encoder for the phrase embeddings, which is more analogous to how BERT handles other language tasks. We see a huge drop in performance. We also try independent BERT encoders (i.e. unshared parameters) between phrase and question embedding models, and we also see a large drop as well. These seem to indicate that a careful design consideration for even small details are crucial when finetuning BERT. Our ablation that excludes coherency scalar decreases DENSPI's EM score by 2% and F1 by 0.2%. This agrees with our intuition that the coherency scalar is useful for precisely defining valid phrase constituents.

6.2 Open-domain Experiments

In this subsection, we evaluate our model's performance (accuracy and speed) on Open-domain SQuAD (SQuAD-Open), which is an extension of SQuAD (Rajpurkar et al., 2016) by Chen et al.

---

[5]The gap is due to constraining the scoring function to be decomposable into question encoder and context encoder.

	F1	EM	s/Q	#D/Q
DrQA	-	29.8	35	5
R^3	37.5	-	-	-
Paragraph ranker	-	30.2	-	20
Multi-step reasoner	39.2	31.9	-	-
MINIMAL	42.5	34.7	-	10
BERTserini	46.1	38.6	115	-
DENSPI	42.3	33.4	0.51	840
−Sparse vector	20.5	13.3	0.22	-
+Pipeline search	38.6	31.2	0.23	-

Table 2: Results on SQuAD-Open (Chen et al., 2017). Top rows are previous models that re-encode documents for every question. The bottom rows are our proposed model. 's/Q' is seconds per query on a CPU and '#/Q' is the number of documents visited per query.

(2017). In this setup, the evidence is the entire English Wikipedia, and the golden paragraphs are not provided for questions.

Model details For the dense vector, we adopt the same setup from Section 6.1 except that we train with no-answer questions (Section 5.1) and an increased batch size of 18. For the sparse vector of each phrase, we use the identical 2-gram tf-idf vector used by Chen et al. (2017), whose vocabulary size is approximately 17 million, of the document that contains the phrase. Since the sparse vector and the dense vector are independently obtained, we tuned the linear scale between the sparse and the dense vectors and found that 0.1 (multiplied on sparse vector) gives the best performance. As discussed in Section 5.3, we perform dense search first; we retrieve top 1000 phrases from the index and rescore the dense vectors with the corresponding sparse (document) vectors.

Baselines We compare our system with previous state-of-the-art models for open-domain question answering. The baselines include DrQA (Chen et al., 2017), MINIMAL (Min et al., 2018), multi-step-reasoner (Das et al., 2019), Paragraph Ranker (Lee et al., 2018), and R^3 (Wang et al., 2018a). We additionally compare with results of a recent paper (Yang et al., 2019) that uses BERT encodings for open-domain QA, and is recently made available on arXiv. We do not experiment with Seo et al. (2018) due to its large gap with DENSPI as demonstrated in Table 1.

Results Table 2 shows the results of our system and previous models on SQuAD-Open. We note following observations: (1) DENSPI outperforms

DrQA by 3.6% EM while achieving 68 times faster inference speed. We previously reported 6K times faster speed in Section 6.1; there is a significant difference largely because DENSPI is covering a larger number of documents than DrQA and we need to account for the overhead during similarity search and disk access (since now most information is on disk). (2) DENSPI is 0.2% F1 behind MINIMAL and 3.8% F1 behind BERTserini, which is BERT ton top of a carefully-engineered paragraph retrieval system. As mentioned in Section 6.1, the difference between ours and BERTserini can be considered as the *decomposability gap* arising from the constraint of query-agnostic phrase representations. We note, however, that the gap is smaller now in open-domain, and the speed-up is 230x.[6] (3) We also report the number of documents that our model computes exact search on and compare it to that of DrQA, as indicated by '#D/Q' in the table. Top-1000 dense search in DENSPI results in 840 unique documents on average, which is much more diverse than the 5 documents that DrQA considers. The benefit of this diversity is better illustrated in the upcoming qualitative analysis (Table 3).

Ablations Table 2 (bottom) shows the effect of the sparse vector and a pipeline search in our method.

Sparse vector: We first try entirely removing the sparse vector, i.e. $\mathbf{x}_{i:j} = \mathbf{d}_{i:j}$ in Equation 1. While this wouldn't have any effect in SQuAD v.1.1, we see a significant drop (-21.8% F1), indicating the importance of the sparse vector in open-domain for distinguishing semantically close but lexically distinct entities.

Pipeline search: The other ablation is using a pipeline search on the sparse (document) vectors first to reduce the search space instead of using dense search (as discussed in Section 5.3), which can be more directly compared to DrQA and other baselines. We see that DENSPI with sparse search first still shows a strong performance with faster inference time (since number of sparse inner product computations decreased to 5), but its accuracy is lower than the original DENSPI and it can explore only few documents.

Qualitative Analysis Table 3 (and Table 5 in Appendix A) contrasts between the results from

---

[6]Assuming BERT-Base processes 130 words per second on CPUs and there are 150 words per paragraph on average.

Q: What can hurt a teacher's mental and physical health? A: occupational stress	
DrQA	[*Mental health*] ... and poor mental health can lead to problems such as **substance abuse**.
DENSPI	[*Teacher*] Teachers face several occupational hazards in their line of work, including **occupational stress**, ...
Q: Who was Kennedy's science adviser that opposed manned spacecraft flights? A: Jerome Wiesner	
DrQA	[*Apollo program*] Kennedy's science advisor **Jerome Wiesner**, (...) his opposition to manned spaceflight ... [*Apollo program*] ... and the sun by NASA manager **Abe Silverstein**, who later said that ... [*Apollo program*] Although Grumman wanted a second unmanned test, **George Low** decided (...) be manned.
DENSPI	[*Apollo program*] Kennedy's science advisor **Jerome Wiesner**, ... his opposition to manned spaceflight ... [*Space Race*] **Jerome Wiesner** of MIT, who served as a (...) advisor to (...) Kennedy, (...) opponent of manned ... [*John F. Kennedy*] ... science advisor **Jerome Wiesner** (...) strongly opposed to manned space exploration, ...
Q: What to do when you're bored?	
DrQA	[*Bored to Death (song)*] I'm nearly bored to **death** [*Waterview Connection*] The twin tunnels were bored by (...) tunnel **boring** machine (TBM) ... [*Bored to Death (song)*] It's easier to say you're bored, or to be angry, than it is to be **sad**.
DENSPI	[*Big Brother 2*] When bored, she enjoys **drawing**. [*Angry Kid*] Angry Kid is (...) bored of long car journeys, so Dad suggests he just **close his eyes and sleep**. [*Pearls Before Swine*] In law school, he became so bored during classes, he started to **doodle a rat**, ...

Table 3: Prediction samples from DrQA and DENSPI in open-domain (English Wikipedia). Each sample shows [*document title*], context, and **predicted answer**.

Q: What was the main radio network in the 1940s in America? A: NBC Red Network	
DENSPI	[*American Broadcasting Company*] In the 1930s, radio in the United States was dominated by (...): the **Columbia Broadcasting System**, the Mutual Broadcasting (...).
Q: Which city is the fifth-largest city in California? A: Fresno	
DENSPI	[*Oakland, California*] **Oakland** is the largest city and the county seat of (...), California, United States.

Table 4: Wrong prediction samples from DENSPI in open-domain (English Wikipedia). Each sample shows [*document title*], context, and **predicted answer**.

model seems to fail to distinguish '1940s' from '1930s'. In the second example, the model seems to focus more on the word 'largest' than the word 'fifth-' in the question.

7 Conclusion

We introduce a model for real-time open-domain question answering by learning indexable phrase representations independent of the query. Our phrase representations leverage sparse and dense vectors to capture lexical, semantic, and syntactic information. On SQuAD-Open, our experiments show that our model can read words 6k times faster under a controlled environment and 68 times faster in a real setup than DrQA while achieving 3.8% higher EM. We believe that even further speedup and larger coverage of documents can be done with a similarity search package for dense+sparse vectors. Future work includes better phrase representation learning to close its accuracy gap with QA models with query-dependent document encoding. Utilizing the phrase index as an external memory for an interaction with text-based knowledge is also an interesting direction.

Acknowledgement

DrQA and DENSPI. In the top example, we note that DrQA fails to retrieve the right document, whereas DENSPI finds the correct answer. This happens exactly because the document retrieval model would not precisely know what kind of content is in the document, while dense search allows it to consider the content directly through phrase-level retrieval. In the second example, while both obtain the correct top-1, DENSPI also obtains the same answer from three different documents. The last example (not from SQuAD) does not have a noun entity, in which a term-frequency-based search engine often performs poorly. We indeed see that DrQA fails because wrong documents are retrieved. On the other hand, DENSPI is able to obtain good answers from several different documents. These results also reinforce the importance of exploring diverse documents ('#D/Q' in Table 2).

Error Analysis Table 4 shows wrong predictions from DENSPI. In the first example, the

References

Alexandr Andoni and Ilya Razenshteyn. 2015. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*.

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *JMLR*.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*.

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

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*.

Jennifer Chu-Carroll, James Fan, BK Boguraev, David Carmel, Dafna Sheinwald, and Chris Welty. 2012. Finding needles in the haystack: Search and candidate generation. *IBM Journal of Research and Development*.

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

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *TPAMI*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*.

Jinhyuk Lee, Seongjun Yun, Hyunjae Kim, Miyoung Ko, and Jaewoo Kang. 2018. Ranking paragraphs for improving answer recall in open-domain question answering. In *EMNLP*.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. In *CoNLL*.

Mike Lewis and Angela Fan. 2019. Generative question answering: Learning to answer the whole question. In *ICLR*.

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

Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *TPAMI*.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*.

Sewon Min, Victor Zhong, Richard Socher, and Caiming Xiong. 2018. Efficient and robust question answering from minimal context over documents. In *ACL*.

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 *CIKM*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL-HLT*.

John Prager et al. 2007. Open-domain question–answering. *Foundations and Trends® in Information Retrieval*.

Jonathan Raiman and John Miller. 2017. Globally normalized reader. In *EMNLP*.

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

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

Minjoon Seo, Tom Kwiatkowski, Ankur Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2018. Phrase-indexed question answering: A new challenge for scalable document comprehension. In *EMNLP*.

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

Anshumali Shrivastava and Ping Li. 2014. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *NIPS*.

Ellen M Voorhees and Dawn M Tice. 2000. Building a question answering test collection. In *SIGIR*.

Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerry Tesauro, Bowen Zhou, and Jing Jiang. 2018a. R 3: Reinforced ranker-reader for open-domain question answering. In *AAAI*.

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

Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-end open-domain question answering with bertserini. *arXiv preprint arXiv:1902.01718*.

Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *EMNLP*.

A More Prediction Samples

Q: Who became the King of the Canary Islands?	
A: Bethencourt	

DrQA	[*Canary Islands*] ... **Winston Churchill** prepared plans (...) of the Canary Islands ...
	[*Isleno* ... In 1501, **Nicols de Ovando** left the Canary Islands ...
	[*Canary Islands*] ... over by **Fernando Clavijo**, the current President of the Canary Islands ...
DENSPI	[*Tenerife*] In 1464, **Diego Garcia de Herrera**, Lord of the Canary Islands, ...
	[*Bettencourt*] ... explorer **Jean de Bthencourt**, who conquered the Canary Islands ...
	[*Bettencourt*] ... **Jean de Bthencourt**, organized an expedition to conquer the Canary Islands, ...

Q: When was the outbreak of World War I?	
A: August 1914	

DrQA	[*Australian Army during World War II*] ... following the outbreak of war in **1939** and ...
	[*Australian Army during World War II*] ... The result was that when war came in **1939**, ...
	[*Australian Army during World War II*] ... the outbreak of the Korean War on 25 June **1950** ...
DENSPI	[*SMS Kaiser Friedrich III*] ... the outbreak of World War I in **July 1914**.
	[*Germany at the Summer Olympics*] At the outbreak of World War I in **1914**, organization ...
	[*Carl Hans Lody*] ... outbreak of the First World War on **28 July 1914** resulted in ...

Q: What comedian is also a university graduate?	
A: Mike Nichols	

DrQA	[*Anaheim University*] ... winning actress and comedian **Carol Burnett** in memory ...
	[*Kettering University*] **Bob Kagle** (...) is one of the most successful venture capitalists ...
	[*Kettering University*] **Edward Davies** (...) is the father-in-law of Mitt Romney.
DENSPI	[*University of Washington*] ... and actor and comedian **Joel McHale** (1995, MFA 2000).
	[*Michigan State University*] ... Fawcett; comedian **Dick Martin, comedian Jackie Martling** ...
	[*West Virginia State University*] ... a comedy show by famed comedian, **Dick Gregory**.

Q: Who is parodied on programs such as Saturday Night Live and The Simpsons?	
A: Doctor Who fandom	

DrQA	[*The Last Voyage of the Starship Enterprise*] ... the "Saturday Night Live" parody of "Star Trek" with **William Shatner**, ...
	[*Saturday Night Live*] ... "Saturday Night Live with **Howard Cosell**" on the rival network ...
	[*Fox Broadcasting Company*] ... "The Late Show", which was hosted by comedian **Joan Rivers**.
DENSPI	[*Gilda Radner*] ... and "Baba Wawa", a parody of **Barbara Walters**.
	[*This American Life*] ... Armisen parodied **Ira Glass** for a skit on "Saturday Night Live"s ...
	[*Anton Chigurh*]... **Chigurh** has been parodied in other media, mainly as a spoof ...

Table 5: More prediction samples from DrQA and DENSPI. Each sample shows [*document title*], context, and **predicted answer**.