Contextualized Sparse Representations for Real-Time Open-Domain Question Answering

Jinhyuk Lee[1] **Minjoon Seo**[2,3] **Hannaneh Hajishirzi**[2,4] **Jaewoo Kang**[1]
Korea University[1] University of Washington[2]
Clova AI, NAVER[3] Allen Institute for AI[4]
{jinhyuk_lee,kangj}@korea.ac.kr
{minjoon,hannaneh}@cs.washington.edu

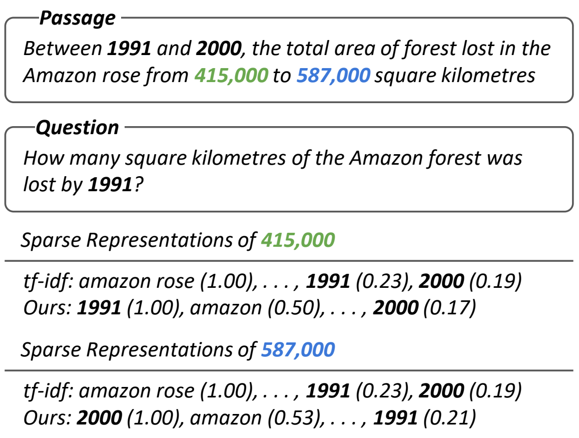
Abstract

Open-domain question answering can be formulated as a phrase retrieval problem, in which we can expect huge scalability and speed benefit but often suffer from low accuracy due to the limitation of existing phrase representation models. In this paper, we aim to improve the quality of each phrase embedding by augmenting it with a contextualized sparse representation (SPARC). Unlike previous sparse vectors that are term-frequency-based (e.g., tf-idf) or directly learned (only few thousand dimensions), we leverage rectified self-attention to indirectly learn sparse vectors in n-gram vocabulary space. By augmenting the previous phrase retrieval model (Seo et al., 2019) with SPARC, we show 4%+ improvement in CuratedTREC and SQuAD-Open. Our CuratedTREC score is even better than the best known *retrieve & read* model with at least 45x faster inference speed.[1]

1 Introduction

Open-domain question answering (QA) is the task of answering generic factoid questions by looking up a large knowledge source, typically unstructured text corpora such as Wikipedia, and finding the answer text segment (Chen et al., 2017). One widely adopted strategy to handle such large corpus is to use an efficient document (or paragraph) retrieval technique to obtain a few relevant documents, and then use an accurate (yet expensive) QA model to *read* the retrieved documents and find the answer (Chen et al., 2017; Wang et al., 2018; Das et al., 2019; Yang et al., 2019).

More recently, an alternative approach formulates the task as an end-to-end phrase retrieval problem by encoding and indexing every possible text span in a dense vector offline (Seo et al., 2018). The approach promises a massive speed advantage with several orders of magnitude lower time complexity, but it performs poorly on entity-centric questions, often unable to disambiguate similar but different entities such as "*1991*" and "*2001*" in dense vector space. To alleviate this issue, Seo et al. (2019) concatenate a term-frequency-based sparse vector with the dense vector to capture lexical information. However, such sparse vector is identical across the document (or paragraph), which means every word's importance is equally considered regardless of its context (Figure 1).



Passage
*Between **1991** and **2000**, the total area of forest lost in the Amazon rose from 415,000 to 587,000 square kilometres*

Question
*How many square kilometres of the Amazon forest was lost by **1991**?*

Sparse Representations of 415,000
*tf-idf: amazon rose (1.00), . . . , **1991** (0.23), **2000** (0.19)*
*Ours: **1991** (1.00), amazon (0.50), . . . , **2000** (0.17)*

Sparse Representations of 587,000
*tf-idf: amazon rose (1.00), . . . , **1991** (0.23), **2000** (0.19)*
*Ours: **2000** (1.00), amazon (0.53), . . . , **1991** (0.21)*

Figure 1: An example of sparse vectors given a context from SQuAD. While *tf-idf* has high weights on infrequent n-grams, our contextualized sparse representation (SPARC) focuses on sematically related n-grams.

In this paper, we introduce a method to learn a Contextualized Sparse Representation (SPARC) for each phrase and show its effectiveness in open-domain QA under phrase retrieval setup. Related previous work (for a different task) often directly maps dense vectors to a sparse vector space (Faruqui et al., 2015; Subramanian et al., 2018), which can be at most only a few thousand dimensions due to computational cost and small gradients. We instead leverage rectified self-attention weights on the neighboring n-grams to scale up its cardinality to n-gram vocabulary space (billions),

[1]Code available at https://github.com/jhyuklee/sparc.

allowing us to encode rich lexical information in each sparse vector. We *kernelize*[2] the inner product space during training to avoid explicit mapping and obtain memory- and computational efficiency.

SPARC improves the previous phrase retrieval model, DenSPI (Seo et al., 2019) (by augmenting its phrase embedding), by more than 4% in both CuratedTREC and SQuAD-Open. In fact, our CuratedTREC result achieves the new state of the art even when compared to previous retrieve & read approaches, with at least 45x faster speed.

2 Background

We focus on open-domain QA on unstructured text where the answer is a text span in a textual corpus (e.g., Wikipedia). Formally, given a set of K documents $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^K$ and a question $\boldsymbol{q}$, the task is to design a model that obtains the answer $\hat{\boldsymbol{a}}$ by $\hat{\boldsymbol{a}} = \text{argmax}_{\boldsymbol{x}^k_{i:j}} F(\boldsymbol{x}^k_{i:j}, \boldsymbol{q})$, where F is the score model to learn and $\boldsymbol{x}^k_{i:j}$ is a phrase consisting of words from the i-th to the j-th word in the k-th document. Pipeline-based methods (Chen et al., 2017; Lin et al., 2018; Wang et al., 2019) typically leverage a document retriever to reduce the number of documents to read, but they suffer from error propagation when wrong documents are retrieved and can be still slow due to the heavy reader model.

Phrase-Indexed Open-domain QA As an alternative, Seo et al. (2018, 2019) introduce an end-to-end, real-time open-domain QA approach to directly encode all phrases in documents agnostic of the question, and then perform similarity search on the encoded phrases. This is feasible by decomposing the scoring function F into two functions,

$$\hat{a} = \text{argmax}_{\boldsymbol{x}^k_{i:j}} H_{\boldsymbol{x}}(\boldsymbol{x}^k_{i:j}) \cdot H_{\boldsymbol{q}}(\boldsymbol{q})$$

where $H_{\boldsymbol{x}}$ is the query-agnostic phrase encoding, and $H_{\boldsymbol{q}}$ is the question encoding, and $\cdot$ denotes a fast inner product operation.

Seo et al. (2019) propose to encode each phrase (and question) with the concatenation of a dense vector obtained via a deep contextualized word representation model (Devlin et al., 2019) and a sparse vector obtained via computing the tf-idf of the document (paragraph) that the phrase belongs to. We argue that the inherent characteristics of tf-idf, which is not learned and identical across the same document, has limited representational power.

[2]Our method is inspired by the kernel method in SVMs (Cortes and Vapnik, 1995).

Our goal in this paper is to propose a better and learned sparse representation model that can further improve the QA accuracy in the phrase retrieval setup.

3 Sparse Encoding of Phrases

Our sparse model, unlike pre-computed sparse embeddings such as tf-idf, *dynamically* computes the weight of each n-gram that depends on the context.

3.1 Why do we need sparse representations?

To answer the question in Figure 1, the model should know that the target answer (***415,000***) corresponds to the year *1991* while the (confusing) phrase *587,000* corresponds to the year *2000*. The dense phrase encoding is likely to have difficulty in precisely differentiating between *1991* and *2000* since it needs to also encode several different kinds of information. Window-based tf-idf would not help because the year *2000* is closer (in word distance) to ***415,000***. This example illustrates the strong need to create an n-gram-based sparse encoding that is highly syntax- and context-aware.

3.2 Contextualized Sparse Representations

The sparse representation of each phrase is obtained as the concatenation of its start word's and end word's sparse embedding, i.e. $\mathbf{s}_{i:j} = [\mathbf{s}^{\text{start}}_i, \mathbf{s}^{\text{end}}_j]$. This way, similarly to how the dense phrase embedding is obtained in Seo et al. (2019), we can efficiently compute them without explicitly enumerating all possible phrases.

We obtain each (start/end) sparse embedding in the same way (with unshared parameters), so we just describe how we obtain the start sparse embedding here and omit the superscript 'start'. Given the contextualized encoding of each document $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_N] \in \mathbb{R}^{N \times d}$, we obtain its (start/end) sparse encoding $\mathbf{S} = [\mathbf{s}_1, \ldots, \mathbf{s}_N] \in \mathbb{R}^{N \times F}$ by

$$\mathbf{S} = \text{ReLU}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{F} \in \mathbb{R}^{N \times F} \quad (1)$$

where $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{N \times d}$ are query, key matrices obtained by applying a (different) linear transformation on $\mathbf{H}$ (i.e., using $W_{\mathbf{Q}}, W_{\mathbf{K}} : \mathbb{R}^{N \times d} \rightarrow \mathbb{R}^{N \times d}$), and $\mathbf{F} \in \mathbb{R}^{N \times F}$ is an one-hot n-gram feature representation of the input document $\boldsymbol{x}$. That is, for instance, if we want to encode unigram (1-gram) features, $\mathbf{F}_i$ will be a one-hot representation of the word x_i, and F will be equivalent to the vocabulary size. Intuitively, $\mathbf{s}_i$ contains a weighted bag-of-ngram representation where each n-gram is weighted by its relative importance on each start or end word of a phrase. Note that F will be very large, so it should always exist as an efficient sparse matrix format (e.g., csc), and one should not explicitly create its dense form. Since we want to handle several different sizes of n-grams, we create the sparse encoding $\mathbf{S}$ for each n-gram and concatenate the resulting sparse encodings. In practice, we experimentally find that unigram and bigram are sufficient for most use cases.

We compute sparse encodings on the question side ($\mathbf{s}' \in \mathbb{R}^F$) in a similar way to the document side, with the only difference that we use the `[CLS]` token instead of start and end words to represent the entire question. We share the same BERT and linear transformation weights used for the phrase encoding.

3.3 Training

As training phrase encoders on the whole Wikipedia is computationally prohibitive, we use training examples from an extractive question answering dataset (SQuAD) to train our encoders. We also use an improved negative sampling method which makes both dense and sparse representations more robust to noisy texts.

Kernel Function Given a pair of question $\boldsymbol{q}$ and a golden document $\boldsymbol{x}$ (a paragraph in the case of SQuAD), we first compute the dense logit of each phrase $\boldsymbol{x}_{i:j}$ by $l_{i,j} = \mathbf{h}_{i:j} \cdot \mathbf{h}'$. Each phrase's sparse embedding is trained, so it needs to be considered in the loss function. We define the sparse logit for phrase $\boldsymbol{x}_{i:j}$ as $l_{i,j}^{\text{sparse}} = \mathbf{s}_{i:j} \cdot \mathbf{s}'_{\texttt{[CLS]}} = \mathbf{s}_i^{\text{start}} \cdot \mathbf{s}'^{\text{start}}_{\texttt{[CLS]}} + \mathbf{s}_j^{\text{end}} \cdot \mathbf{s}'^{\text{end}}_{\texttt{[CLS]}}$. For brevity, we describe how we compute the first term $\mathbf{s}_i^{\text{start}} \cdot \mathbf{s}'^{\text{start}}_{\texttt{[CLS]}}$ corresponding to the start word (and dropping the superscript 'start'); the second term can be computed in the same way.

$$\mathbf{s}_i^{\text{start}} \cdot \mathbf{s}'^{\text{start}}_{\texttt{[CLS]}} = \\ \text{ReLU}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)_i \mathbf{F} \quad \left(\text{ReLU}\left(\frac{\mathbf{Q}'\mathbf{K}'^\top}{\sqrt{d}}\right)_{\texttt{[CLS]}} \mathbf{F}'\right)^\top \tag{2}$$

where $\mathbf{Q}', \mathbf{K}' \in \mathbb{R}^{M \times d}, \mathbf{F}' \in \mathbb{R}^{M \times F}$ denote the question side query, key, and n-gram feature matrices, respectively. The output size of $\mathbf{F}$ is prohibitively large, but we efficiently compute the loss by precomputing $\mathbf{F}\mathbf{F}'^\top \in \mathbb{R}^{N \times M}$. Note that $\mathbf{F}\mathbf{F}'^\top$ can be considered as applying a kernel function, i.e. $K(\mathbf{F}, \mathbf{F}') = \mathbf{F}\mathbf{F}'^\top$ where its (i, j)-th entry is 1 if and only if the n-gram at the i-th position of the context is equivalent to the j-th n-gram of the question, which can be efficiently computed as well. One can also think of this as *kernel trick* (in the literature of SVM (Cortes and Vapnik, 1995)) that allows us to compute the loss without explicit mapping.

The final loss to minimize is computed from the negative log likelihood over the sum of the dense and sparse logits:

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

where i^*, j^* denote the true start and end positions of the answer phrase. As we don't want to sacrifice the quality of dense representations which is also very critical in dense-first search explained in Section 4.1, we add dense-only loss that omits the sparse logits (i.e. original loss in Seo et al. (2019)) to the final loss, in which case we find that we obtain higher-quality dense phrase representations.

Negative Sampling To learn robust phrase representations, we concatenate negative paragraphs to the original SQuAD paragraphs. To each paragraph $\boldsymbol{x}$, we concatenate the paragraph $\boldsymbol{x}_{\text{neg}}$ which was paired with the question whose dense representation $\mathbf{h}'_{\text{neg}}$ is most similar to the original dense question representation $\mathbf{h}'$, following Seo et al. (2019). We find that adding tf-idf matching scores on the word-level logits of the negative paragraphs further improves the quality of sparse representations.

4 Experiments

4.1 Experimental Setup

Datasets SQuAD-Open is the open-domain version of SQuAD (Rajpurkar et al., 2016). We use 87,599 examples with the golden evidence paragraph to train our encoders and use 10,570 examples from dev set to test our model, as suggested by Chen et al. (2017). CuratedTrec consists of question-answer pairs from TREC QA (Voorhees et al., 1999) curated by Baudiš and Šedivỳ (2015). We use 694 test set QA pairs for testing our model. We only train on SQuAD and test on both SQuAD-Open and CuratedTREC, relying on the generalization ability of our model (zero-shot) for CuratedTREC.

Model	C.TREC EM	SQuAD-Open EM	SQuAD-Open F1	s/Q
Models with Dedicated Search Engines				
DrQA	25.4*	29.8**	-	35
R^3	28.4*	29.1	37.5	-
Paragraph Ranker	**35.4***	30.2	-	161
Multi-Step-Reasoner	-	31.9	39.2	-
BERTserini	-	38.6	46.1	115
Multi-passage BERT	-	**53.0**	**60.9**	84
End-to-End Models				
ORQA	30.1	20.2	-	8.0
DENSPI	31.6†	36.2	44.4	0.71
DENSPI + SPARC (Ours)	**35.7†**	**40.7**	**49.0**	0.78

* Trained on distantly supervised training data.
** Trained on multiple datasets
† No supervision using target training data.

Table 1: Results on two open-domain QA datasets. See Appendix A for how s/Q is computed.

Implementation Details We use and finetune BERT-Large for our encoders. We use BERT vocabulary which has 30,522 unique tokens based on byte pair encodings. As a result, we have $F \approx 1B$ when using both uni-/bigram features. We do not finetune the word embedding during training. We pre-compute and store all encoded phrase representations of all documents in Wikipedia (more than 5 million documents). It takes 600 GPU hours to index all phrases in Wikipedia. We use the same storage reduction and search techniques by Seo et al. (2019). For search, we perform dense search first and then rerank with sparse scores (DFS) or perform sparse search first and rerank with dense scores (SFS), or a combination of both (Hybrid).

Comparisons For models using dedicated search engines, we show performances of DrQA (Chen et al., 2017), R^3 (Wang et al., 2018), Paragraph Ranker (Lee et al., 2018), Multi-Step-Reasoner (Das et al., 2019), BERTserini (Yang et al., 2019), and Multi-passage BERT (Wang et al., 2019). For end-to-end models that do not rely on search engine results, DENSPI (Seo et al., 2019), ORQA (Lee et al., 2019), and DENSPI + SPARC (Ours) are evaluated. For DENSPI and ours, 'Hybrid' search strategy is used.

4.2 Results

Open-Domain QA Experiments Table 1 shows experimental results on two open-domain question answering datasets, comparing our method with previous pipeline and end-to-end approaches. On both datasets, our model with contextualized sparse representations (DENSPI + SPARC) largely improves the performance of the phrase-indexing baseline model (DENSPI) by more than 4%. Also, our method runs significantly faster than other models that need to run heavy QA models during the inference. On CuratedTREC, which is constructed from real user queries, our model achieves state-of-the-art performance at the time of submission. Even though our model is only trained on SQuAD (i.e., zero-shot), it outperforms all other models which are either distant- or semi-supervised with at least 45x faster inference.

On SQuAD-Open, our model outperforms BERT-based pipeline approaches such as BERTserini (Yang et al., 2019) while being more than two orders of magnitude faster. Multi-passage BERT, which utilizes a dedicated document retriever, outperforms all end-to-end models with a large margin in SQuAD-Open. While our main contribution is on the improvement in end-to-end, we also note that retrieving correct documents in SQuAD-Open is known to be often easily exploitable (Lee et al., 2019), so we should use more open-domain-appropriate test datasets (such as CuratedTREC) for a more fair comparison.

Model	$\text{SQuAD}_{1/100}$	$\text{SQuAD}_{1/10}$
Ours	60.0	51.6
− SPARC	55.9 (−4.1)	48.4 (−3.2)
− Doc./Para. tf-idf	58.1 (−1.9)	50.9 (−0.7)
+ Trigram SPARC	58.0 (−2.0)	49.8 (−1.8)

Table 2: Ablations of our model. We show effects of different sparse representations.

	Model	EM	F1
Original	DrQA (Chen et al., 2017)	69.5	78.8
	BERT (Devlin et al., 2019)	84.1	90.9
Query-Agnostic	LSTM + SA + ELMo	52.7	62.7
	DENSPI	73.6	81.7
	DENSPI + SPARC	76.4	84.8

Table 3: Results on the SQuAD development set. LSTM+SA+ELMo is a query-agnostic baseline from Seo et al. (2018).

Ablation Study Table 2 shows the effect of contextualized sparse representations by comparing different variants of our method on SQuAD-Open. We use a subset of Wikipedia dump (1/100 and 1/10). Interestingly, adding trigram features in SPARC is worse than using uni-/bigram representations only, calling for a stronger regularization for

Q: When is Independence Day?	
DrQA	[*Independence Day (1996 film)*] Independence Day is a **1996** American science fiction ...
DENSPI + SPARC	[*Independence Day (India)*] ... is annually observed on **15 August** as a national holiday in India.
Q: What was the GDP of South Korea in 1950?	
DRQA	[*Economy of South Korea*] In 1980, the South Korean GDP per capita was **$2,300**.
DENSPI	[*Economy of South Korea*] In 1980, the South Korean GDP per capita was **$2,300**.
DENSPI + SPARC	[*Developmental State*] South Korea's GDP grew from **$876** in 1950 to $22,151 in 2010.

Table 4: Prediction samples from DrQA, DENSPI, and DENSPI + SPARC. Each sample shows [*document title*], context, and **predicted answer**.

high-order n-gram features. See Appendix B on how SPARC performs in different search strategies.

Closing the Decomposability Gap Table 3 shows the performance of DenSPI + SPARC in the SQuAD v1.1 development set, where a single paragraph that contains an answer is provided in each sample. While BERT-Large that jointly encodes a passage and a question still has a higher performance than ours, we have closed the gap to 6.1 F1 score in a query-agnostic setting.

Qualitative Analysis Table 4 shows the outputs of three OpenQA models: DrQA (Chen et al., 2017), DENSPI (Seo et al., 2019), and DENSPI + SPARC (ours). Our model is able to retrieve various correct answers from different documents, and it often correctly answers questions with specific dates or numbers compared to DENSPI showing the effectiveness of learned sparse representations.

5 Conclusion

In this paper, we demonstrate the effectiveness of contextualized sparse representations, SPARC, for encoding phrase with rich lexical information in open-domain question answering. We efficiently train our sparse representations by kernelizing the sparse inner product space. Experimental results show that our fast open-domain QA model that augments DENSPI with SPARC outperforms previous open-domain QA models, including recent BERT-based pipeline models, with two orders of magnitude faster inference time.

Acknowledgments

This research was supported by National Research Foundation of Korea (NRF-2017R1A2A1A17069 645, NRF-2017M3C4A7065887), ONR N00014-18-1-2826, DARPA N66001-19-2-403, Allen Distinguished Investigator Award, and Sloan Fellowship. We thank the members of Korea University, University of Washington, NAVER Clova AI, and the anonymous reviewers for their insightful comments.

References

Petr Baudiš and Jan Šedivỳ. 2015. Modeling of the question answering task in the yodaqa system. In *CLEF*.

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

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning*.

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

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

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A Smith. 2015. Sparse overcomplete word vector representations. In *ACL-IJCNLP*.

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*.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *ACL*.

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

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

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*.

Minjoon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur P Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2019. Real-time open-domain question answering with dense-sparse phrase index. In *ACL*.

Anant Subramanian, Danish Pruthi, Harsh Jhamtani, Taylor Berg-Kirkpatrick, and Eduard Hovy. 2018. Spine: Sparse interpretable neural embeddings. In *AAAI*.

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

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

Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. 2019. Multi-passage bert: A globally normalized bert model for open-domain question answering. In *EMNLP*.

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. In *NAACL Demo*.

A Inference Speed Benchmark of Open-Domain QA Models

Table 6 shows how the inference speed of each open-domain QA model is estimated in our benchmarks. Many of these models are not open-sourced but based on BERT, so we use the speed of BERT on the given length token as the basis for computing the inference speed.

We also note that our reported number for the inference speed of DenSPI (Seo et al., 2019) is slightly faster than that reported in the original paper. This is mostly because we are using a PCIe-based SSD (NVMe) instead of SATA-based. We also expect that the speed-up can be greater with Intel Optane which has faster random access time.

B Model Performances in Different Search Strategies

Model	SQuAD-Open		CuratedTREC	
	DENSPI	+ SPARC	DENSPI	+ SPARC
SFS	33.3	36.9 (+3.6)	28.8	30.0 (+1.2)
DFS	28.5	34.4 (+5.9)	29.5	34.3 (+4.8)
HYBRID	36.2	40.7 (+4.5)	31.6	35.7 (+4.1)

Table 5: Exact match scores of SPARC in different search strategies. SFS: Sparse First Search. DFS: Dense First Search. HYBRID: Combination of SFS + DFS. Exact match scores are reported.

In Table 5, we show how we consistently improve over DENSPI when SPARC is added in different search strategies. Note that on CuratedTREC where the questions more resemble real user queries, DFS outperforms SFS showing the effectiveness of dense search when not knowing which documents to read.

Models using Bi-LSTM as Base Encoders

Model	# of Docs. to Read	# of Docs. to Re-rank	s/Q
DrQA	5	None	35
Paragraph Ranker	≈ 3	20	161

Models using BERT as Base Encoders

Model	# of Docs. to Read	# of Docs. to Re-rank	Max. Sequence Length/Doc. Stride	s/Q
BertSerini	100 paragraphs*	100 paragraphs*	384/128	115
Multi-Passage BERT	30 passages**	100 passages**	100/50	84
ORQA	k blocks†	None	384/128	8

* Assumed 1 paragraph = 200 BERT tokens.
** 1 passage = 100 BERT tokens.
† 1 block = 288 BERT tokens. Assumed $k = 10$.

Table 6: Details on how the inference speed of each open-domain QA model is estimated using a single CPU. We use the default 384/128 (Max. Sequence Length/Doc. Stride) setting for both BertSerini and ORQA.