Effective and Efficient Query-aware Snippet Extraction for Web Search

Jingwei Yi[1], **Fangzhao Wu**[2], **Chuhan Wu**[3], **Xiaolong Huang**[4], **Binxing Jiao**[4], **Guangzhong Sun**[1], **Xing Xie**[2]
[1]University of Science and Technology of China [2]Microsoft Research Asia
[3]Tsinghua University [4]Microsoft STC Asia
yjw1029@mail.ustc.edu.cn {wufangzhao,wuchuhan15}@gmail.com
{xiaolhu,binxjia,xingx}@microsoft.com gzsun@ustc.edu.cn

Abstract

Query-aware webpage snippet extraction is widely used in search engines to help users better understand the content of the returned webpages before clicking. Although important, it is very rarely studied. In this paper, we propose an effective query-aware webpage snippet extraction method named DeepQSE, aiming to select a few sentences which can best summarize the webpage content in the context of input query. DeepQSE first learns query-aware sentence representations for each sentence to capture the fine-grained relevance between query and sentence, and then learns document-aware query-sentence relevance representations for snippet extraction. Since the query and each sentence are jointly modeled in DeepQSE, its online inference may be slow. Thus, we further propose an efficient version of DeepQSE, named Efficient-DeepQSE, which can significantly improve the inference speed of DeepQSE without affecting its performance. The core idea of Efficient-DeepQSE is to decompose the query-aware snippet extraction task into two stages, i.e., a coarse-grained candidate sentence selection stage where sentence representations can be cached, and a fine-grained relevance modeling stage. Experiments on two real-world datasets validate the effectiveness and efficiency of our methods.

1 Introduction

Given an input search query, search engines such as Google [1] and Bing [2], not only return the URLs and the titles of the relevant webpages, but also show the query-aware snippets of these webpages, aiming to help users better understand the webpage content before clicking. These webpage snippets are usually one or two sentences extracted from the webpage, which can not only summarize the key content of the webpage, but also be relevant to the input query. Some examples are shown in Figure 1. For the query 'einstein' and the webpage of 'Albert Einstein - Wikipedia', a good snippet is a brief introduction of Einstein's life. While for the query 'einstein achievement', a good snippet would be sentences describing his influence on science. In other words, the snippet is a summarization of the webpage in the context of input query.

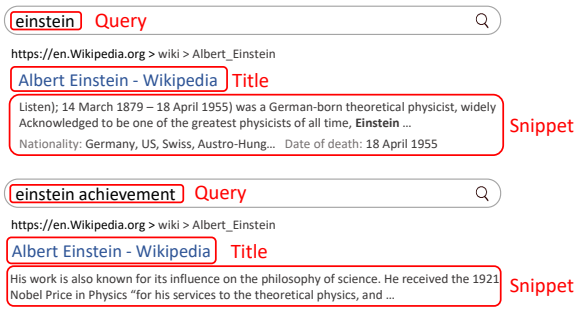


Figure 1: Examples of query-aware snippets in search engines.

Although query-aware webpage snippet extraction is important and useful, it is very rarely studied. Only a few works exist in this field, and most of them are based on simple word-level text matching method (Penin et al., 2008; Zou et al., 2021). For example, Turpin et al. (2007) proposed to utilize the number of overlapping words between queries and sentences in webpages to extract snippets. Tsegay et al. (2009) proposed to select snippets through the summation of Kullback-Leibler divergence or TF-IDF weight of overlapping words between queries and sentences in webpages. However, these methods rely on counting features of overlapping words, and cannot capture the deep semantic relation between query and webpage.

In this paper, we propose an effective query-aware webpage snippet extraction method for web search, named DeepQSE [3]. In DeepQSE, given an input query and a webpage with multiple sentences,

[1]https://www.google.com
[2]https://www.bing.com

[3]https://github.com/yjw1029/DeepQSE.

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 3035 - 3046
December 7-11, 2022

we first learn query-aware sentence representations for each sentence to capture the fine-grained relevance between query, sentence and webpage title using a query-aware sentence encoder. Then we model the query-sentence relevance in the context of the whole webpage using a document-aware relevance encoder. Since the query and each webpage sentence are jointly modeled, the online inference speed of DeepQSE can be slow, while the search engines have extremely high requirements for low latency. Thus, we further design an efficient version of DeepQSE named Efficient-DeepQSE, aiming to significantly improve the inference speed of DeepQSE and keep its performance as much as possible. The key idea of Efficient-DeepQSE is to decompose the query-aware webpage snippet extraction task into two stages, i.e., coarse-grained candidate sentence selection and fine-grained relevance modeling. The coarse-grained candidate sentence selection aims to select a moderate number of most potential sentences for snippet extraction using a bi-encoder where sentence representations can be cached for fast online serving. The fine-grained relevance modeling stage aims to capture the deep semantic relevance between the query and the candidate sentences selected by the previous stage using query-aware cross-encoders. We conducted many experiments on two real-world datasets, which verify the effectiveness and efficiency of our approach. The contributions of this paper are as follows:

- We propose an effective query-aware webpage snippet extraction method for web search named DeepQSE, which can summarize the webpage content in the context of input query.
- We further propose Efficient-DeepQSE which can improve the inference speed of Deep-QSE with a minor performance drop.
- We conduct extensive experiments on two real-world datasets to verify the effectiveness and efficiency of our methods.

2 Related Work

2.1 Query-aware Snippet Extraction

Query-aware snippet extraction is a widely-used technique to select snippets which can help users better understand the webpage content before clicking (Chen et al., 2020; Bando et al., 2010). Although important, only a few works have been proposed for query-aware snippet extraction based on word-level text matching method (Zou et al., 2021; Turpin et al., 2007; Penin et al., 2008; Tsegay et al., 2009). For example, Turpin et al. (2007) propose CTS, which selects snippets based on sentence positions and the number of overlapping words between queries and sentences. Zou et al. (2021) propose QUITE, which computes importance scores for each word and sums the importance scores of overlapping words to select snippets. These methods are mostly based on counting features of overlapping words and cannot capture deep semantic relations between query and webpage. Recently, Zhong et al. (2021) propose QMSUM for meeting summarization, of which the locator can be used for snippet extraction. The locator of QMSUM applies a fixed PLM and CNN to encode sentence and query, and a Transformer to model interactions between sentences. QMSUM is a bi-encoder which fails to encoder the word-level interactions between query and sentences. Zhao et al. (2021) propose QBSUM, which concatenates query and body, and applies multiple predictors to compute relevance scores. The simple body-query concatenation in QBSUM may fluctuate the information of query and lead to some sentences being cut off due to the length limitation of PLM. Recently, some works (Ishigaki et al., 2020; Chen et al., 2020) use abstractive generation model to generate snippets for (query, document) pairs. For example, Ishigaki et al. (2020) uses the RNN network with copy mechanism to generate query-aware snippets. However, abstractive methods need detailed parsing and digesting, which usually takes a considerable amount of time (Wang et al., 2007). Therefore, these methods are not compared in this paper.

2.2 Text Matching

Text matching has been widely applied in many scenarios, such as information retrieval (Pang et al., 2017) and clustering various articles for breaking news detection (Yang et al., 2002). Recently several text matching methods have been proposed. Following Humeau et al. (2020), these methods can be divided into two groups, i.e., bi-encoders and cross-encoders. Bi-encoders (Palangi et al., 2014; Reimers and Gurevych, 2019; Hu et al., 2014) model the sentence-level interactions between queries and documents, in which the document representations can be cached for fast online serving. For example, Wan et al. (2016) propose C-DSSM, which computes query and document representations with convolutional networks.

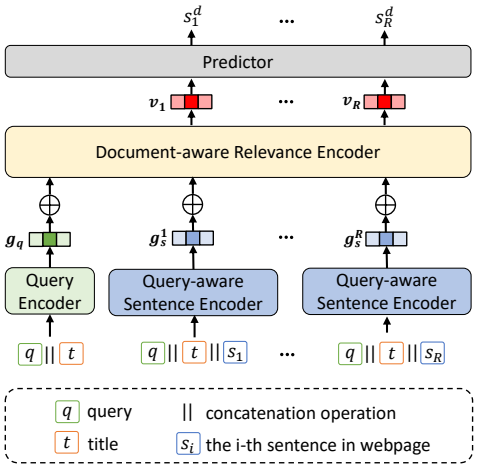


Figure 2: Architecture of DeepQSE.

Cross-encoders (Guo et al., 2016; Li et al., 2020; Chen et al., 2018) model the word-level interactions between queries and documents in a fine-grained manner. For example, Yilmaz et al. (2019) propose Birch, where (title, query) pairs are input into a pre-trained language model to compute matching scores. Cross-encoders usually perform better than bi-encoders (Urbanek et al., 2019), but have higher computation overhead since they cannot cache the document representations. Since text matching methods can retrieve the most relevant sentence to query, we treat them as baseline methods and compare the performance with them in Section 4. However, the text matching methods only consider the similarity between queries and sentences, and ignore the contextual information of webpages, which might be sub-optimal.

3 Methodology

In this section, we give the problem formulation of query-aware snippet extraction. Then we introduce our DeepQSE and Efficient-DeepQSE in detail.

3.1 Problem Formulation

When a user submits a request with query q, the search engine returns several webpages. Given one of the webpage d with title t, it contains several sentences $\{s_1, s_2, ...s_R\}$, where R is the max number of sentences in a webpage. The snippet extraction model aims to select several consecutive sentences $\{s_k, ...s_{k+n}\}$ as the snippet that can summarize the webpage content in the context of input query. Since the number of sentences n is given by the pre-defined snippet length, the snippet extraction model needs to select the start sentence s_k.

3.2 DeepQSE

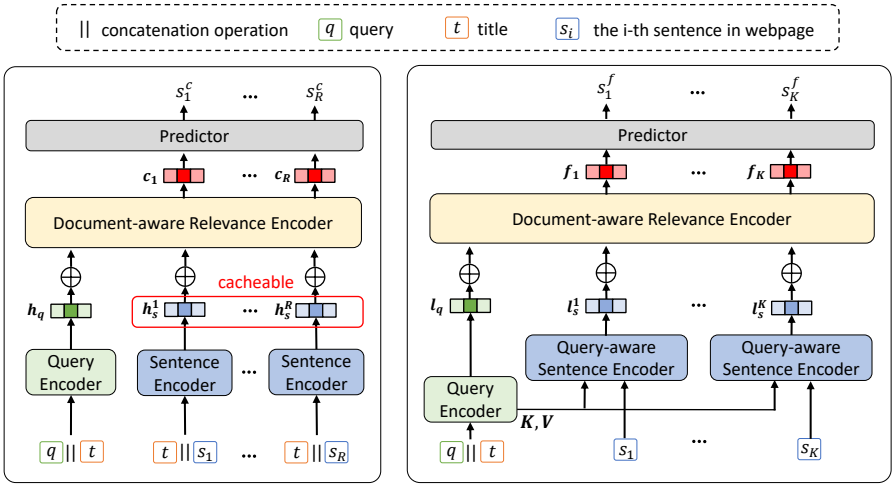
DeepQSE aims to select snippets which can best summarize the webpage content in the context of the input query. The model structure of DeepQSE is shown in Figure 2, which is composed of a query encoder, a query-aware sentence encoder and a document-aware relevance encoder. The query encoder learns query representations, which is initialized from a pre-trained language model, such as BERT (Devlin et al., 2018) and XML-RoBERTa (Chi et al., 2021). Given the query q and title t, the concatenation of them is input into the query encoder. The final hidden state of the first token is the query representation $\mathbf{g}_q$. The query-aware sentence encoder models the word-level interactions between query, title and each sentence to compute query-aware sentence representations. It is initialized from a pre-trained language model, of which the input is the concatenation of title, query and each sentence. The final hidden state of the first token is the sentence representation $\mathbf{g}_s^i$. The document-aware relevance encoder aims to model the sentence-level interactions between the query and sentences in the context of the whole webpage, which is composed of several Transformer blocks (Vaswani et al., 2017). We concatenate the query representation and sentence representations, add position embeddings and input them into the document-aware relevance encoder. The final hidden states are used as document-aware query-sentence relevance representations $\mathbf{v}_i$, which are then used to compute the selection score s_i^d.

3.3 Efficient-DeepQSE

In DeepQSE, the query and each sentence are jointly modeled, which may have slow computation speed for online serving. In order to reduce the computation overhead, we further design an efficient version of DeepQSE named Efficient-DeepQSE, which is shown in Figure 3. We decompose the query-aware snippet extraction into two stages, i.e., coarse-grained candidate sentence selection and fine-grained relevance modeling.

3.3.1 Coarse-grained Sentence Selection

The coarse-grained candidate sentence selection aims to select K candidate sentences and parse them to the fine-grained relevance model for final snippet extraction. It separates the modeling of candidate sentences and queries, which enables caching sentence representations for fast online serving. The model structure of the coarse-grained



(a) Coarse-grained candidate sentence selection (b) Fine-grained relevance modeling

Figure 3: Framework of the two stages in Efficient-DeepQSE.

candidate selector is shown in Figure 3(a), which contains three core modules, i.e., a query encoder, a sentence encoder and a document-aware relevance encoder. The query encoder and sentence encoder aim to learn the query and sentence representations respectively, which are initialized from a pre-trained language model. We input the concatenation of query and title into the query encoder, and use the final hidden states of the first token as the query representation $\mathbf{h}_q$. The concatenation of title and each sentence is input into the sentence encoder, and the final hidden states of the first token are treated as the sentence representation $\mathbf{h}_s^i$. The document-aware sentence relevance encoder aims to model query-sentence relevance in the context of the whole webpage, which is composed of several Transformer blocks. We concatenate the query representation and sentence representations, add position embeddings and input them into the document-aware sentence relevance encoder. The final hidden states are the document-aware query-sentence relevance representations $\mathbf{c}_i$, which are further used to predict selection scores s_i^c through an MLP network.

3.3.2 Fine-grained Relevance Modeling

The fine-grained relevance modeling aims to capture the deep semantic relevance between query and the candidate sentences parsed from the coarse-grained sentence selection stage. It is composed of a query encoder, a query-aware sentence encoder and a document-aware relevance encoder, of which the model structure is shown in Figure 3(b). A naive implementation is directly using the same architecture of DeepQSE. However, in DeepQSE, the query and title are concatenated with different R sentences and their word representations are repetitively computed for R times. We assume the word representations of query in the query-aware sentence encoder have little help for query-aware snippet selection, which is validated in Section 4.6. Therefore, we design the Cross Transformer, where the word representations of the query and title are only computed once in the query encoder and parsed into the query-aware sentence encoder. The architecture of a Cross Transformer block is shown in Figure 4.

The query encoder aims to learn query representations, which is initialized from a Transformer-based pre-trained language model. We input the concatenation of query and title into the query encoder and use the final hidden state of the first token as the query representation $\mathbf{l}_q$. Meanwhile, the query encoder outputs the key and value of every multi-head self attention network to the query-aware sentence encoder. Given the previous hidden state $\mathbf{H}_q^{i-1}$, the key and value of the i-th multi-head self attention network are computed as follows:

$$\mathbf{K}_i^q = \mathbf{W}_K^i \mathbf{H}_q^{i-1}, \mathbf{V}_i^q = \mathbf{W}_V^i \mathbf{H}_q^{i-1}, \quad (1)$$

where $\mathbf{W}_K^i$ and $\mathbf{W}_V^i$ are trainable parameters.

The query-aware sentence encoder aims to model the fine-grained interactions between query and each sentence. It contains an embedding layer and several Cross Transformer blocks, which are initialized from a pre-trained language model. Given a sentence s_i, we first compute its initial hidden state $\mathbf{H}_s^0$ through the embedding layer. The i-th

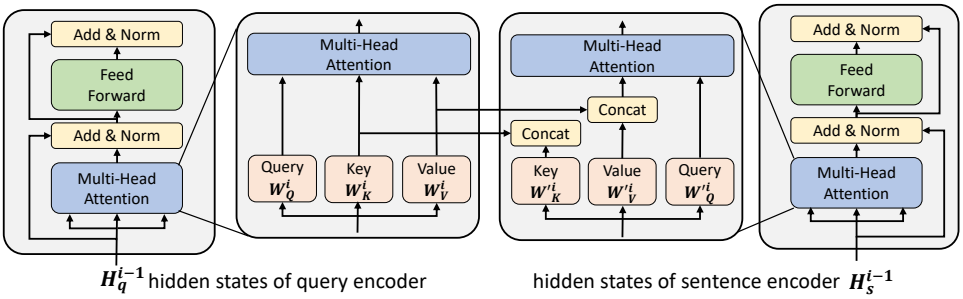


Figure 4: Architecture of the Cross Transformer model.

Cross Transformer block contains a multi-head attention network and a feed-forward network. In order to compute query-aware sentence hidden states, we modify the key (or value) of multi-head attention network as the concatenation of key (or value) from query encoder and the transformed hidden state of the sentence. Given the hidden state $\mathbf{H}_s^{i-1}$ of the previous Cross Transformer block, the query, key and value of the i-th multi-head attention network are formulated as follows:

$$\begin{aligned} \mathbf{Q}_i^s &= \mathbf{W}_Q^{i'}\mathbf{H}_s^{i-1}, \\ \mathbf{K}_i^s &= \mathbf{K}_i^q || \mathbf{W}_K^{i'}\mathbf{H}_s^{i-1}, \\ \mathbf{V}_i^s &= \mathbf{V}_i^q || \mathbf{W}_V^{i'}\mathbf{H}_s^{i-1}, \end{aligned} \tag{2}$$

where $\mathbf{W}_Q^{i'}$, $\mathbf{W}_K^{i'}$ and $\mathbf{W}_V^{i'}$ are trainable parameters, $||$ is the concatenation operator. The query, key and value are then input into the multi-head attention network and feed-forward network to compute $\mathbf{H}_s^i$. We use the final hidden state of the first token as the query-aware sentence representation $\mathbf{l}_s^i$.

The document-aware relevance encoder aims to model query-sentence relevance in the context of the whole webpage, which contains several Transformer blocks. We concatenate the query representations and K candidate sentence representations, add position embeddings on them and input them into several transformer blocks. The document-aware query-sentence relevance representations $\mathbf{f}_i$ are the final hidden states, which are then fed into an MLP to predict selection scores s_i^f.

3.3.3 Model Training and Serving

For model training, we use cross-entropy loss to train the DeepQSE, which is computed as follows:

$$\mathcal{L} = -\sum_{i=1}^{R} y_i \times log(\frac{exp(s_i^d)}{\sum_{k=1}^{R} exp(s_k^d)}), \tag{3}$$

where $y_i \in \{0, 1\}$ indicates whether the i-th sentence is the start sentence of the snippet. For Efficient-DeepQSE, we also use the cross-entropy loss to train the coarse-grained candidate sentence selector and the fine-grained relevance model respectively, which is formulated as follows:

$$\begin{aligned} \mathcal{L}_c &= -\sum_{i=1}^{R} y_i \times log(\frac{exp(s_i^c)}{\sum_{k=1}^{R} exp(s_k^c)}), \\ \mathcal{L}_f &= -\sum_{i=1}^{R} y_i \times log(\frac{exp(s_i^f)}{\sum_{k=1}^{R} exp(s_k^f)}). \end{aligned} \tag{4}$$

For model serving, when a user submits a request with query q, the search engine returns several webpages. For one of the webpages d with title t, DeepQSE directly computes selection scores for all sentences $\{s_1^d, ...s_R^d\}$. The sentence with the maximum selection score is selected. For Efficient-DeepQSE, the server needs to offline compute the sentence representations of coarse-grained candidate sentence selector for every webpage. For the webpage d, with its sentence representations of the coarse-grained candidate sentence selector $[\mathbf{h}_s^1, ...\mathbf{h}_s^R]$, we first compute the query representation of the coarse-grained candidate sentence selector $\mathbf{h}_q$ and the coarse-grained selection scores $\{s_1^c, ...s_R^c\}$. Then we feed top-K candidate sentences into the fine-grained relevance model to compute fine-grained selection scores $\{s_1^f, ...s_K^f\}$. The sentence with the maximum score is the start sentence of the snippet s_k.

Snippet extraction dataset	*English*	*Multi-lingual*
#sample	545,240	348,673
#query	420,816	291,559
#document	330,545	240,005
Manually labeled dataset	*English*	*Multi-lingual*
#sample	19,331	25,851
#query	14,522	17,935
#document	16,995	22,726

Table 1: Statistics of datasets.

Methods	*English*			*Multi-lingual*		
	P@1	P@3	P@5	P@1	P@3	P@5
CTS	39.65±0.00	64.57±0.00	88.15±0.00	34.64±0.00	59.83±0.00	71.16±0.00
QUITE	39.49±0.00	63.69±0.00	74.78±0.00	33.71±0.00	57.24±0.00	68.96±0.00
QMSUM	54.22±0.27	74.61±0.20	82.17±0.24	46.26±0.27	67.19±0.18	75.87±0.17
QBSUM	60.69±0.51	77.18±0.26	81.12±0.38	59.61±0.77	71.80±0.99	74.84±0.67
BM25	33.91±0.00	60.50±0.00	73.15±0.00	27.75±0.00	51.99±0.00	65.48±0.00
DSSM	40.24±0.50	59.98±0.59	69.12±0.54	36.88±0.30	54.33±0.34	63.29±0.29
C-DSSM	55.46±0.31	72.73±0.23	79.39±0.26	49.49±0.41	66.32±0.37	73.80±0.30
MatchPyramid	55.49±0.40	78.04±0.40	85.78±0.26	50.74±0.35	73.58±0.30	82.05±0.39
Poly-Encoder	64.45±0.11	82.19±0.08	88.02±0.11	64.41±0.13	82.14±0.10	88.00±0.07
Birch	72.45±0.08	88.24±0.10	92.73±0.08	72.62±0.11	88.44±0.14	92.88±0.06
PARADE	73.19±0.16	88.65±0.12	92.99±0.14	72.94±0.20	88.47±0.13	92.89±0.13
DeepQSE*	**77.05**±0.29	**92.43**±0.33	95.94±0.14	**77.23**±0.18	**93.30**±0.09	**96.77**±0.08
Efficient-DeepQSE*	**77.03**±0.27	**91.98**±0.19	95.34±0.13	**75.13**±0.27	**91.40**±0.22	**95.15**±0.14

Table 2: Performance of different methods on query-aware webpage snippet extraction.

4 Experiments

4.1 Dataset and Experimental Settings

Since there is no off-the-shelf dataset for query-aware snippet extraction, we first manually labeled two small *English* and *Multi-lingual* datasets, of which the task is to select the more proper snippet given a pair of candidate sentences. The *Multi-lingual* dataset includes 10 languages, i.e., German, French, Spanish, Italian, Japanese, Korean, Portuguese, Russian and Chinese. Then we semi-automatically build two large *English* and *Multi-lingual* snippet extraction datasets with part of the manually-labeled datasets, of which task is to select the snippet from the sentences in the body. Due to the space limitation, the detailed dataset construction steps are described in Appendix 6. 10% samples of the snippet extraction dataset are randomly sampled for testing, and the rest for training. We randomly sample 10% samples of the training dataset for validation. We also the rest manually labeled dataset as another test dataset, which is not used to construct the large snippet extraction dataset. The detailed statistics of the datasets are shown in Table 1. We use precision@k (k=1,3,5) as the evaluation metrics for performance on the snippet extraction test dataset, accuracy (ACC) as the evaluation metric for performance on the manually labeled test dataset, floating-point operations (FLOPs) and million seconds (ms) as the evaluation metrics for efficiency.

4.2 Experimental Settings

In our experiments, we apply BERT-base (Devlin et al., 2018) for *English* dataset and a distilled XML-RoBERTa (Chi et al., 2021) for *Multi-Lingual* dataset to initialize the pre-trained language model. We use Adam (Kingma and Ba, 2015) to optimize model training for both DeepQSE and Efficient-DeepQSE. We set the learning rate as 0.0001 and batch size as 64. The maximum query length is 16. The maximum title length is 32. The maximum sentence length is 64. The maximum number of sentences R in a body is 160. The number of candidate sentences selected by the coarse-grained sentence selector K is 20. All hyper-parameters are selected according to the performance on the validation dataset. We repeat each experiment 5 times and report the average results and the standard deviations.

4.3 Performance Comparison

We compare our method with multiple baselines, including conventional snippet extraction methods: (1) CTS (Turpin et al., 2007), extracting snippets based on the number of overlapping words between queries and sentences; (2) QUITE (Zou et al., 2021), selecting snippets with the summation of importance scores of overlapping words between queries and sentences; PLM-empowered snippet extraction methods: (3) QMSUM (Zhong et al., 2021), the locator of QMSUM for meeting summarization which applies a fixed PLM and CNN to encode sentence and query, and a Transformer to model interactions between sentences. (4) QBSUM (Zhao et al., 2021), input the concatenation of query and body into a PLM, and apply multiple predictors to compute relevance scores; some text matching methods: (5) BM25 (Robertson and Zaragoza, 2009), applying the BM25 algorithm to compute similarity scores; (6) DSSM (Huang

Methods	ACC	
	English	*Multi-lingual*
CTS	29.20±0.00	29.84±0.00
QUITE	24.82±0.00	25.73±0.00
QMSUM	38.05±0.16	39.61±0.21
QBSUM	21.73±0.86	26.26±0.65
BM25	33.10±0.00	33.11±0.00
DSSM	36.67±0.46	37.16±0.18
C-DSSM	36.97±0.26	37.77±0.21
MatchPyramid	35.54±0.16	38.28±0.16
Poly-Encoder	37.51±0.23	39.82±0.22
Birch	38.17±0.28	39.83±0.21
PARADE	36.68±0.17	38.59±0.13
DeepQSE*	**40.10**±0.58	**41.99**±0.20
Efficient-DeepQSE*	**40.57**±0.34	**41.99**±0.26

Table 3: Performance of different methods on manually labeled datasets.

Methods	*English*		*Multi-lingual*	
	FLOPs	ms	FLOPS	ms
CTS	-	1.10	-	1.51
QUITE	-	0.13	-	0.10
QBSUM	45.75G	7.97	11.40G	4.02
QMSUM	1.74G	0.41	0.43G	0.17
BM25	-	0.80	-	0.95
DSSM	0.30M	0.18	0.30M	0.09
C-DSSM	17.42M	0.17	17.42M	0.09
MatchPyramid	0.28G	0.37	0.28G	0.38
Poly-Encoder	1.55G	0.19	0.43G	0.08
Birch	1087.44G	21.72	271.86G	10.51
PARADE	1088.71G	22.35	272.17G	11.16
DeepQSE*	1540.08G	31.91	271.86G	16.70
Efficient-DeepQSE*	132.45G	3.09	33.44G	1.67

Table 4: Efficiency of different methods.

et al., 2013), a deep structured semantic matching method; (7) C-DSSM (Wan et al., 2016), a deep semantic matching structure with convolution network; (8) MatchPyramid (Pang et al., 2016), applying 2D convolution and max-pooling network on the similarity matrix of query and sentence; several PLM-empowered text matching methods: (9) Poly-Encoder (Humeau et al., 2020), which adds a final attention mechanism to model the interactions between the cacheable multiple sentence representations and the query representation. (10) Birch (Yilmaz et al., 2019), inputting the concatenation of queries and sentences into BERT for document retrieval; (11) PARADE (Li et al., 2020), using a PLM to model similarity between sentences and queries, and an aggregator to model interactions between candidate sentences.

The performance of all methods on snippet extraction test datasets is shown in Table 2. The performance of the methods on manually labeled test datasets is shown in Table 3. CTS, QUITE and BM25 are deterministic methods, of which standard deviations are zero. We have several observations from Table 2. First, our DeepQSE and Efficient-DeepQSE outperform conventional snippet extraction methods (CTS and QUITE). This is because these methods are based on the counting features of overlapping words between queries and sentences. Compared with our methods which utilize PLMs, they cannot capture the deep semantic relation between query and sentences. Second, our methods outperform PLM-based snippet extraction methods (QMSUM and QBSUM). This is because the simple body-query concatenation in QBSUM may fluctuate the information of the short query. Due to the length limitation of PLM some candidate sentences may be cut off. QMSUM is a bi-encoder which fails to encoder the word-level interactions between the query and sentences. Third, compared with several text matching methods (BM25, DSSM, C-DSSM, MatchPyramid, Poly-Encoder, Birch, QBSUM), our methods have better performance. This is because in our methods we apply webpage title and document-aware relevance encoder to select snippets in the context of the whole webpage, which can choose sentences better summarizing the webpage in the context of input query. Forth, PLM-based snippet extraction methods outperform conventional snippet extraction methods, and PLM-based text-matching methods outperform shallow-model-based text-matching methods. This is because the pre-trained language model can help better understand the semantic information in queries, titles and sentences. Finally, cross-encoder-based text matching methods outperform bi-encoder-based text matching methods. For example, MatchPyramid outperforms CDSSM and DSSM, and Birch, PARADE and DeepQSE outperform Poly-Encoder and QBSUM. This is because bi-encoders only model sentence-level similarity between queries and sentences, but cross-encoders can model word-level similarity between queries and sentences in a fine-grained manner. However, bi-encoders can cache sentence representations, which have faster online serving speed than cross-encoders.

4.4 Efficiency Comparison

In this subsection, we compare the efficiency of DeepQSE and Efficient-DeepQSE with baseline methods. The results are summarized in Table 4.

Since CTS, QUITE and BM25 are not based on matrix multiplication and addition, we do not give their FLOPs results. We have several observations from Table 4. First, conventional snippet extraction methods (CTS and QUITE) have relatively low computation costs. This is because they are based on simple hand-crafted features, which can be calculated quickly. Second, cross-encoder-based methods are more time-consuming than bi-encoder-based methods. For example, DSSM and CDSSM are more efficient than MatchPyramid, and Poly-Encoder and QMSUM are more efficient than Birch, PARADE, QBSUM and DeepQSE. This is because the sentence representations in bi-encoder-based methods can be cached for quick inference. Third, PLM-based methods (Poly-Encoder, Birch, PARADE, QMSUM, QBSUM, DeepQSE and Efficient-DeepQSE) have higher computation costs than other methods. This is because pre-trained language models have large size of parameters, of which the computation cost is high (Sanh et al., 2019; Beltagy et al., 2020; Jiao et al., 2020; Sun et al., 2020). Finally, considering both efficiency and the previous performance analysis in Section 4.3, our Efficient-DeepQSE achieves a great trade-off between performance and efficiency. This is because our Efficient-DeepQSE applies two-stage model, in which the coarse-grained selector can quickly select candidates for the fine-grained relevance encoder. In addition, we design the Cross Transformer which avoids repetitively computing contextual word representations of the same query for different candidate sentences. Therefore, our Efficient-DeepQSE has a lower computation cost while keeping its performance.

4.5 Efficiency Analysis

In this subsection, we analyze how the Efficient-DeepQSE reduces the computation overhead of DeepQSE with a minor performance drop. Compared with DeepQSE, the core improvement of Efficient-DeepQSE is the Cross Transformer, the coarse-grained candidate sentence selector and the fine-grained relevance model. We remove these modules separately and show their performance and efficiency in Figure 5, Figure 6 and Figure 7. We have several observations from the results. First, Efficient-DeepQSE has lower performance and lower computation overhead without the fine-grained relevance model. This is because the fine-grained relevance model captures the deep semantic relevance between queries, titles and sentences, which can improve the performance. And without the fine-grained relevance model, Efficient-DeepQSE does not need to perform the second stage, which lowers the computation overhead. Second, Efficient-DeepQSE can achieve higher performance but higher computation overhead without the coarse-grained candidate sentence selector. This is because the coarse-grained candidate sentence selector may select candidate sentences incorrectly, which increases the error rate. However, it helps decrease the input size of the second stage. Therefore, the computation overhead gets higher without the coarse-grained candidate sentence selector. Finally, the computation overhead is higher without Cross Transformer. This is because in Cross Transformer we only compute the query and title representations once, which avoids the repetitive computation in DeepQSE. Combined with these components, the Efficient-DeepQSE reduces the computation overhead and achieves comparable performance with DeepQSE.

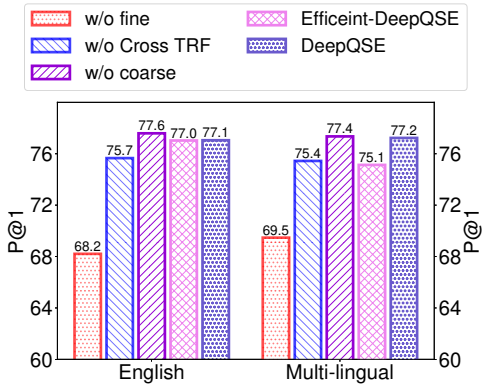


Figure 5: The impact of two-stage model and Cross Transformer on accuracy.

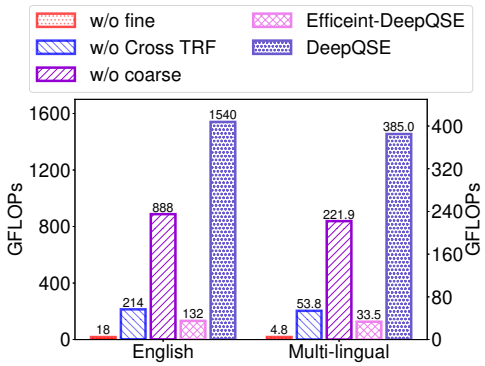


Figure 6: The impact of two-stage model and Cross Transformer on efficiency (GFLOPs).

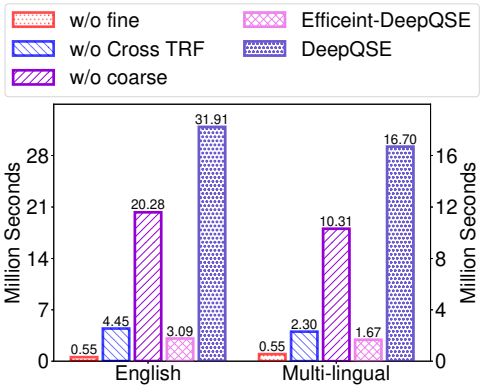


Figure 7: The impact of two-stage model and Cross Transformer on efficiency (ms).

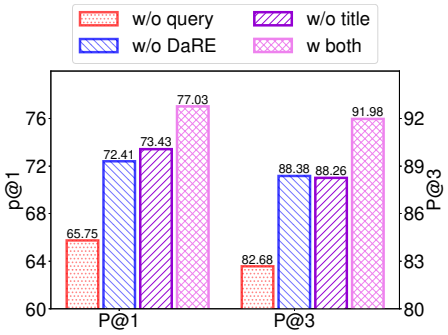


Figure 8: The impact of title, query and document-aware relevance encoder (DaRE) on *English* dataset.

4.6 Ablation Study

In this section, we analyze the impact of adding titles, queries and the document-aware relevance encoder. Due to space limitation, we only show the experimental result on Efficient-DeepQSE. The same phenomenon can be observed on DeepQSE as well. The results are shown in Figure 8 and Figure 9. From the results, we can observe that the performance of Efficient-DeepQSE gets lower without titles. This is because the titles can be treated as brief abstracts of webpages, which can help the model select sentences better summarizing the webpage content (Wang et al., 2007). It is also observed that the performance drops without queries. This is because the selected snippets should not only summarize the webpage content, but also be relevant to queries. Finally, the document-aware relevance encoder (DaRE) benefits snippet extraction. This may be because the document-aware relevance encoder can model the query-document relevance in the context of the whole webpage, which helps select snippets better summarizing the webpage content.

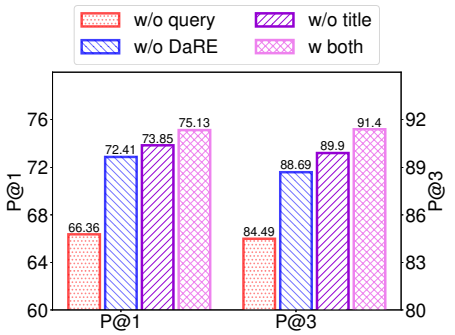


Figure 9: The impact of title, query and document-aware relevance encoder (DaRE) on *Multi-lingual* dataset.

5 Conclusion

In this paper, we propose a query-aware snippet extraction model for web search named DeepQSE. DeepQSE first learns a query-aware sentence representation by modeling fine-grained interactions between queries, titles and sentences, then learns document-aware sentence relevance representations for snippet extraction. To lower the computation overhead of DeepQSE, we further design the Efficient-DeepQSE, where the snippet extraction is decomposed into two stages, i.e. coarse-grained candidate sentence selection and fine-grained relevance modeling. The coarse-grained selector can cache the sentence representations for fast online serving and parse several candidate sentences to the fine-grained relevance model. In the fine-grained relevance model, we further design a Cross Transformer, to avoid the repetitive computation of query and title representations for different sentences. Extensive experiments validate the effectiveness and efficiency of our approach.

6 Limitations

Our DeepQSE is a cross-encoder-based snippet extraction method. It has great performance but heavy computation overhead, which is not beneficial for online inference. We further propose Efficient-DeepQSE, an efficient version of DeepQSE, which divides the snippet extraction into two stages. Although the Efficient-DeepQSE keeps the performance of DeepQSE and has much lower computation overhead than other PLM-based methods, it still has larger computation overhead than the conventional shallow-model-based methods. We plan to further improve the efficiency of the snippet extraction algorithm in the future.

References

Lorena Leal Bando, Falk Scholer, and Andrew Turpin. 2010. Constructing query-biased summaries: A comparison of human and system generated snippets. In *IIiX*, page 195–204.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Haolan Chen, Fred X. Han, Di Niu, Dong Liu, Kunfeng Lai, Chenglin Wu, and Yu Xu. 2018. Mix: Multi-channel information crossing for text matching. In *KDD*, page 110–119.

Wei-Fan Chen, Shahbaz Syed, Benno Stein, Matthias Hagen, and Martin Potthast. 2020. Abstractive snippet generation. In *WWW*, page 1309–1319.

Zewen Chi, Li Dong, Furu Wei, Nan Yang, Saksham Singhal, Wenhui Wang, Xia Song, Xian-Ling Mao, Heyan Huang, and Ming Zhou. 2021. InfoXLM: An information-theoretic framework for cross-lingual language model pre-training. In *ACL*, pages 3576–3588.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186.

Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *CIKM*, pages 55–64.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *NIPS*, page 2042–2050.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, pages 2333–2338.

Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. 2020. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *ICLR*.

Tatsuya Ishigaki, Hen-Hsen Huang, Hiroya Takamura, Hsin-Hsi Chen, and Manabu Okumura. 2020. Neural query-biased abstractive summarization using copying mechanism. In *ECIR*, pages 174–181.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *EMNLP Findings*, pages 4163–4174.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Canjia Li, Andrew Yates, Sean MacAvaney, Ben He, and Yingfei Sun. 2020. Parade: Passage representation aggregation for document reranking. *arXiv preprint arXiv:2008.09093*.

Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and R Ward. 2014. Semantic modelling with long-short-term memory for information retrieval. *arXiv preprint arXiv:1412.6629*.

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2016. Text matching as image recognition. In *AAAI*, page 2793–2799.

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. 2017. Deeprank: A new deep architecture for relevance ranking in information retrieval. In *CIKM*, page 257–266.

Thomas Penin, Haofen Wang, Thanh Tran, and Yong Yu. 2008. Snippet generation for semantic web search engines. In *The Semantic Web*, pages 493–507.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *EMNLP*, pages 3982–3992.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, page 333–389.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *ACL*, pages 2158—2170.

Yohannes Tsegay, Simon J Puglisi, Andrew Turpin, and Justin Zobel. 2009. Document compaction for efficient query biased snippet generation. In *ECIR*, pages 509–520. Springer.

Andrew Turpin, Yohannes Tsegay, David Hawking, and Hugh E. Williams. 2007. Fast generation of result snippets in web search. In *SIGIR*, page 127–134.

Jack Urbanek, Angela Fan, Siddharth Karamcheti, Saachi Jain, Samuel Humeau, Emily Dinan, Tim Rocktäschel, Douwe Kiela, Arthur Szlam, and Jason Weston. 2019. Learning to speak and act in a fantasy text adventure game. In *EMNLP*, pages 673–683.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*, page 6000–6010.

Shengxian Wan, Yanyan Lan, Jiafeng Guo, Jun Xu, Liang Pang, and Xueqi Cheng. 2016. A deep architecture for semantic matching with multiple positional sentence representations. In *AAAI*, page 2835–2841.

Changhu Wang, Feng Jing, Lei Zhang, and Hong-Jiang Zhang. 2007. Learning query-biased web page summarization. In *CIKM*, page 555–562.

Yiming Yang, Jaime Carbonell, Ralf Brown, John Lafferty, Thomas Pierce, and Thomas Ault. 2002. *Multi-strategy Learning for Topic Detection and Tracking*, pages 85–114.

Zeynep Akkalyoncu Yilmaz, Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Cross-domain modeling of sentence-level evidence for document retrieval. In *EMNLP*, pages 3490–3496.

Mingjun Zhao, Shengli Yan, Bang Liu, Xinwang Zhong, Qian Hao, Haolan Chen, Di Niu, Bowei Long, and Weidong Guo. 2021. Qbsum: A large-scale query-based document summarization dataset from real-world applications. *Computer Speech & Language*, 66:101166.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. 2021. QMSum: A new benchmark for query-based multi-domain meeting summarization. In *NAACL*, pages 5905–5921.

Lixin Zou, Shengqiang Zhang, Hengyi Cai, Dehong Ma, Suqi Cheng, Shuaiqiang Wang, Daiting Shi, Zhicong Cheng, and Dawei Yin. 2021. Pre-trained language model based ranking in baidu search. In *KDD*, pages 4014–4022.

query	title	snippet
death park	download death park 1 8 0 for android	Death Park is a first-person horror game that turns you into a young man who's being chased by a dark clown.
Chromatophores in squid	chromatophores gilly lab	Chromatophores in the skin of squid ... each is an elastic pigment body, spherical at rest, surrounded by a halo of...
1 km to 1 mile	1 mi to km converter	And the answer is 0.6213711922 mi in 1 km. ... 1.609344 km in 1 mi.

Figure 10: Some snippets extracted by DeepQSE.

query	title	snippet
what is the size of the keyspace AES	encryption key length and message length in aes ...	AES has variable key sizes like 128,192, and 256, therefore, the keyspace K is 2 128, 2 192, and 2 256...
What are three natural barriers found in Ancient China?	natural barriers ancient china	There are a total of eleven natural barriers surrounding China, these include the Himalayas, Yellow Sea, Mount Everest...
at dead of night	at dead of night free download top pc games	At Dead Of Night is part horror film, part horror game and part ghost hunt. ...

Figure 11: Some snippets extracted by Efficient-DeepQSE.

Appendix

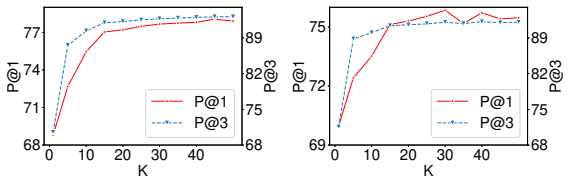
Detailed Data Construction Steps

The detailed data construction steps are as follows:
Collect manually labeled dataset: Given a pair of candidate snippets extracted from a document, human evaluators are asked to select the more appropriate one according to the corresponding query. The manually labeled dataset can be formulated as $\{(q_i, s_{i1}, s_{i2}, d_i, l_i)|0 \leq i < M\}$, where M is the number of samples in the dataset, q_i is the query, s_{i1} and s_{i2} are candidate snippets, d_i is the document, $l_i \in \{0, 1\}$ is the label. l_i equals 0 when s_{i1} is more suitable than s_{i2}, otherwise l_i equals 1. At least three annotators are assigned for a sample.

Build snippet extraction dataset: Since the manually labeled dataset is for pair-wise selection and is small for large PLM-based models, we train an ensemble model with the dataset to extract snippets for different documents according to corresponding queries. The extracted samples with high confidence scores are then used as the snippet extraction dataset.

Impact of Candidate Number

In this subsection, we study the impact of the number of candidate sentences selected by the coarse-grained selector. The experimental results are shown in Figure 12(a) and Figure 12(b). We observe that with larger candidate sentence number, the performance of Efficient-DeepQSE is higher. This may be because the probability that the ground truth sentence is selected in the candidate sentences gets higher. However, the computation overhead of Efficient-DeepQSE linearly increases with larger candidate sentence number. How to choose a proper candidate sentence number to achieve a great trade-off between performance and efficiency is the key point of our method.



(a) *English* (b) *Multi-Lingual*

Figure 12: Impact of candidate number.

Case Study

In this subsection, we show some snippets extracted by our DeepQSE in Figure 10 and our Efficient-DeepQSE in Figure 11. In all cases, the snippets are relevant to the input query. This is because we model the word-level interactions between query and sentences in DeepQSE and Efficient-DeepQSE. Meanwhile, the selected snippets summarize the webpage content in the context of the input query. This is because we consider the context of webpage in the document-aware relevance encoder, which enables our method to capture the global webpage information.

Experimental Environments

We conduct experiments with a Linux server with 8 V100 GPUs with 32GB memory. The version of CUDA is 11.1. We implement both DeepQSE and Efficient-DeepQSE with pytorch 1.9.1.