Generating Keyword Queries for Natural Language Queries to Alleviate Lexical Chasm Problem

Xiaoyu Liu*, Shunda Pan*, Qi Zhang†, Yu-Gang Jiang, Xuanjing Huang
School of Computer Science, Shanghai Key Laboratory of Intelligent Information Processing
Fudan University, Shanghai, P.R.China 201203
{liuxiaoyu16,sdpan17,qz,ygj,xjhuang}@fudan.edu.cn

ABSTRACT

In recent years, the task of reformulating natural language queries has received considerable attention from both industry and academic communities. Because of the lexical chasm problem between natural language queries and web documents, if we directly use natural language queries as inputs for retrieval, the results are usually unsatisfactory. In this work, we formulated the task as a translation problem to convert natural language queries into keyword queries. Since the nature language queries users input are diverse and multi-faceted, general encoder-decoder models cannot effectively handle low-frequency words and out-of-vocabulary words. We propose a novel encoder-decoder method with two decoders: the pointer decoder firstly extracts query terms directly from the source text via copying mechanism, then the generator decoder generates query terms using two attention modules simultaneously considering the source text and extracted query terms. For evaluation and training, we also proposed a semi-automatic method to construct a large-scale dataset about natural language query-keyword query pairs. Experimental results on this dataset demonstrated that our model could achieve better performance than the previous state-of-the-art methods.

CCS CONCEPTS

• **Information systems** → **Query reformulation**; • **Computing methodologies** → *Machine translation*;

KEYWORDS

Query Reformulation; Natural Language Queries; Neural Encoder-Decoder Model; Pointer Network

ACM Reference Format:
Xiaoyu Liu, Shunda Pan, Qi Zhang, Yu-Gang Jiang, Xuanjing Huang. 2018. Generating Keyword Queries for Natural Language Queries to Alleviate Lexical Chasm Problem. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271727

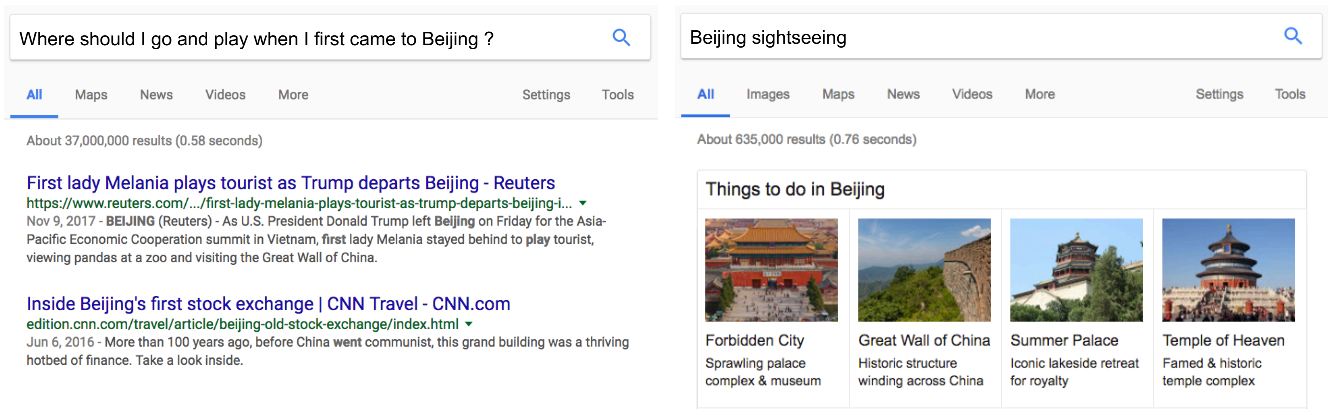
*These authors contributed equally to this work.
†Corresponding authors.

1 INTRODUCTION

Most of the traditional web search engines were designed to process keyword queries. In most cases, they can provide satisfactory retrieval results when a query is composed of only a few keywords. However, along with the development of speech recognition and mobile phones, virtual assistants (e.g. Google Allo or Microsoft Cortana) are more commonly used. Hence, natural language queries have become widely used as inputs for retrieval. In contrast to queries that contain a few keywords, natural language queries not only contain many common words but also suffer from the "lexical chasm" problem [25]. Figure 1 illustrates an example. From this example, we can observe that the input natural language query contains many common words, such as *"where", "go" and "play"*. However, it does not contain important keywords like *"sightseeing"* or *"things to do"*, which are commonly used in the relevant documents. As a result, its retrieval results are unsatisfactory because of the redundant words and lexical chasm problem. However, we will get the satisfactory result if we use the corresponding keyword query to retrieval.

Because of an increasing need, this problem has received considerable attention from both the academic and commercial communities. Query rewrite, which aims to alleviate the vocabulary chasm by altering a given original query into an alternative query, is a major solution for the problem, and various related approaches have been proposed in recent years [8, 17, 25, 30]. For example, Jones et al. [17] proposed the generation of a new query to replace the natural language query and treated query rewriting as a machine translation problem. Since then, both statistical machine translation models [8, 25] and neural machine translation models [30] have been proposed to rewrite natural language queries. However, these models cannot effectively handle low-frequency keywords and those keywords that do not exist in the fixed size output vocabulary. For example, some name entities (e.g. Beijing) only appear in few natural language queries, and generating these words from a fixed-size output vocabulary is difficult.

Another challenge of this task is that there is lack of publicly available datasets about natural language query - keyword query pairs. This significantly impacts the further development of academic research. The existing studies [8, 17, 25, 30] all constructed datasets based on the query logs of commercial search engines (e.g. Google, Yahoo or Bing), which are difficult for the academic community to obtain. Although the Text Retrieval Conference (TREC) conducted an answer retrieval contest in 2017 and released


Where should I go and play when I first came to Beijing ?

About 37,000,000 results (0.58 seconds)

First lady Melania plays tourist as Trump departs Beijing - Reuters
https://www.reuters.com/.../first-lady-melania-plays-tourist-as-trump-departs-beijing-i...
Nov 9, 2017 - **BEIJING** (Reuters) - As U.S. President Donald Trump left **Beijing** on Friday for the Asia-Pacific Economic Cooperation summit in Vietnam, **first** lady Melania stayed behind to **play** tourist, viewing pandas at a zoo and visiting the Great Wall of China.

Inside Beijing's first stock exchange | CNN Travel - CNN.com
edition.cnn.com/travel/article/beijing-old-stock-exchange/index.html
Jun 6, 2016 - More than 100 years ago, before China **went** communist, this grand building was a thriving hotbed of finance. Take a look inside.

Beijing sightseeing

About 635,000 results (0.76 seconds)

Things to do in Beijing

Forbidden City	Great Wall of China	Summer Palace	Temple of Heaven
Sprawling palace complex & museum	Historic structure winding across China	Iconic lakeside retreat for royalty	Famed & historic temple complex

(a) **natural language query** (b) **keyword query**

Figure 1: An example of retrieval results of the natural language query *"Where should I go and play when I first came to Beijing"* and keyword query *"Beijing sightseeing"*. If we can translate the natural language query into its corresponding keyword query, we will get satisfactory retrieval results.

a related dataset [6], there will be a significant gap between this dataset and users' true natural language queries. The natural language queries are often relatively colloquial and unspecialized, but this dataset was based on large quantities of knowledge articles from Wikipedia and its original queries are composed of an article and section title.

To overcome these issues, in this work, we first constructed a large dataset of natural language queries using the largest Chinese Community Question Answering (CQA) platform "Baidu Knows"[1]. This dataset contains a collection of natural language queries and their corresponding keyword queries. Because manual annotation is a time-consuming and expensive task, we proposed a semi-automatic method to perform the annotation. By analyzing the data, we find that the corresponding keyword query is composed of two parts: *extractive keywords*, which appear in the original natural language queries, and *generative keywords*, which do not appear in the original natural language queries. Hence, our proposed model contains two decoders: *pointer decoder* and *generator decoder*. The pointer decoder only extracts extractive keywords directly from the original language query, while the generator decoder can generate generative keywords from a fixed-size output vocabulary. By introducing the pointer decoder, our model could extract those low-frequency words and out-of-output-vocabulary words that appeared in the original query.

In addition, we deliberately designed the process of keyword queries generation with a focus on how real human annotators do it. Given a natural language query, human annotators usually first extract keywords from the original queries, then they may summarize some keywords that do not appear in the original queries relying on an understanding of the original query and extractive keywords. In our model, the pointer decoder first extracts keywords from the source text with relatively high accuracy to remedies the exposure bias issue. Then, the generator decoder summarizes and generates keywords to alleviate lexical chasm using two attention modules simultaneously considering source text and extracted keywords. Experimental results on our dataset showed that our method could achieve better performance in translating natural language queries to keyword queries, and effectively alleviated both the redundant words and lexical chasm problem.

The main contributions of this work are summarized as follows:

- We proposed a semi-automatic method to perform the data annotation and constructed a large-scale dataset about natural language queries for academic research. This dataset[2] will be released with our code together.
- We proposed a novel encoder-decoder method with two decoder, where the pointer decoder firstly extracts keywords directly from the source text via copying mechanism, then the generator decoder generates query terms using two attention modules simultaneously considering the source text and extracted keywords.
- Experimental results demonstrated that the proposed method could achieve better performance than the previous state-of-the-art methods in natural language queries rewrite.

2 RELATED WORK

There are three major areas related to this task and our proposed models. The first is the work on the task of query reformulation in information retrieval. The second is the encoder-decoder model for the translation problem, and the last is the pointer network and copying mechanism. We will mainly introduce related studies in these areas in the following section.

2.1 Query Reformulation in Information Retrieval

Query reformulation has long been an important research topic in information retrieval, and there have been a number of approaches

[1]https://zhidao.baidu.com

[2]https://github.com/fudannlp16/cikm_query_keyword

for this topic. Xu and Croft [34] used the top-ranked returned documents retrieved by the original query to expand the query. This method is influenced by the initial ranking results and can not utilize user-generated data. Later approaches focused on using user query logs to expand a query by means of the clickthrough rate [5, 35], co-occurrence in search sessions [17], or query similarity based on click-through graphs [7]. The advantage of these approaches is that user feedback is readily available in user query logs and can efficiently be precomputed. However, these approaches all face the problem of requiring some resource dependent on a specific search engine. In addition, He et al. [14] proposed a learning to rewrite framework that focuses on the candidate ranking.

More recently, several studies adopted data from user query logs as input for the Statistical Machine Translation (SMT) model for learning query rewriting and query expansion [8, 9, 25, 26]. These methods treated user queries as one language and the reformulated queries as another language. The effectiveness of the statistical translation-based approach to a web search has been demonstrated empirically in recent studies, in which word-based and phrase-based translation models were trained on large amounts of click-through data [1].

Some methods that translate natural language queries into keyword queries have also been proposed. Huston and Croft [16] removed some redundant words, and Kumaran and Giridhar [19] used a predictor model to reduce long queries. Maxwell and Croft [21] selected the words directly from a query according to the importance calculated by their models. These methods focused on extracting keywords directly from the original query, but could not generate keywords that were not available in the original query. Song et al. [30] proposed a neural attentional encoder-decoder model to translate natural language queries into keyword queries, to generate keywords that were not seen in the original queries. However, this method could not effectively handle low-frequency words and out-of-output-vocabulary words

2.2 Encoder-Decoder Model

Some previous works formulated the task of translating natural language queries into keyword queries as a translation problem. In this work, we also regarded it as a translation problem and adopted the encoder-decoder model.

Cho et al. [3] proposed to the use of an RNN encoder-decoder that consisted of two recurrent neural networks for machine translation. The model encoded a source sentence into a fixed-length vector, from which a decoder generated a translation. An attention mechanism was first introduced into a translation models by Bahdanau et al [2] and later refined by Luong et al. [20] and others. The key idea behind the attention mechanism is to establish direct shortcut connections between the target and the source by paying attention to relevant source content when we translate. Encoder-decoder models with an attention mechanism have already been widely used in several NLP tasks such as machine translation [2, 20], speech recognition [10], and text summarization [4, 27].

2.3 Pointer Network

Our idea of introducing a pointer decoder into the attentional encoder-decoder model was partially inspired by the recent work of Pointer Networks [32], in which an encoder-decoder model used the soft attention distribution of Bahdanau et al [2] to predict the output sequence directly from the input sequence. The pointer network has been used to create hybrid approaches that mix pointing (copying) and generation during decoding in various tasks like machine translation [12], language modeling [22], and summarization [11, 24, 29]. In addition, He et al. [13] recently proposed a method to generate natural answers for real-world question answering systems by incorporating copying and retrieving mechanisms into the encoder-decoder model.

Our model is close to the CopyNet model of Gu et al. [11]. and the Pointer-Generator Networks model of See et al [29]. However, we considered using two different decoders to copy and generate keywords separately, while their models adopted just one decoder to copy or generate words at each translation step. In our model, the pointer decoder first extracts keywords from the source text with relatively high accuracy to remedies the exposure bias issue. Then, the generator decoder summarizes and generates new keywords to alleviate lexical chasm using two attention modules simultaneously considering source text and extracted keywords.

3 METHODOLOGY

In this section, we will describe our proposed novel encoder-decoder model in details. As illustrated in Figure 2, our encoder-decoder framework mainly includes three components: **1)** a neural **Encoder *(e)*** to encode the natural language query into hidden state representations; **2)** a neural **Pointer Decoder (p)** to extract *extractive keywords* directly from the natural language query; and **3)** a neural **Generator Decoder (g)** to generate *generative keywords* from a fixed-size output vocabulary.

3.1 Problem Definition

Given a dataset that consists of N data samples, the i-th data sample $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ contains one natural language query $\mathbf{x}^{(i)}$ and its corresponding keyword query $\mathbf{y}^{(i)}$. The natural language query $\mathbf{x}^{(i)}$ is a word sequence, while the keyword query $\mathbf{y}^{(i)}$ is a word set. To apply the encoder-decoder model, the word set $\mathbf{y}^{(i)}$ needs to be converted into a word sequence. In particular, we sorted $\mathbf{y}^{(i)}$ in descending order according to the each word's TF-IDF score, and regarded the ordered $\mathbf{y}^{(i)}$ as a word sequence. This sorting process was conducted in the data annotation (section 4.2). As a consequence, both $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$ are sequences of words:

$$\mathbf{x}^{(i)} = x_1^{(i)}, x_2^{(i)}, ..., x_{L_{\mathbf{x}^{(i)}}}^{(i)}$$

$$\mathbf{y}^{(i)} = y_1^{(i)}, y_2^{(i)}, ..., y_{L_{\mathbf{y}^{(i)}}}^{(i)}$$

where $L_{\mathbf{x}^{(i)}}$ and $L_{\mathbf{y}^{(i)}}$ represents the length of word sequence of $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$, respectively.

When a natural language query $\mathbf{x}^{(i)}$ is given, the goal of our model is to translate $\mathbf{x}^{(i)}$ into its corresponding keyword query $\mathbf{y}^{(i)}$. By analyzing the data, we find that 45% words of $\mathbf{y}$ are only related to the semantic meaning of $\mathbf{x}$, instead of appearing in the $\mathbf{x}$. Meanwhile,

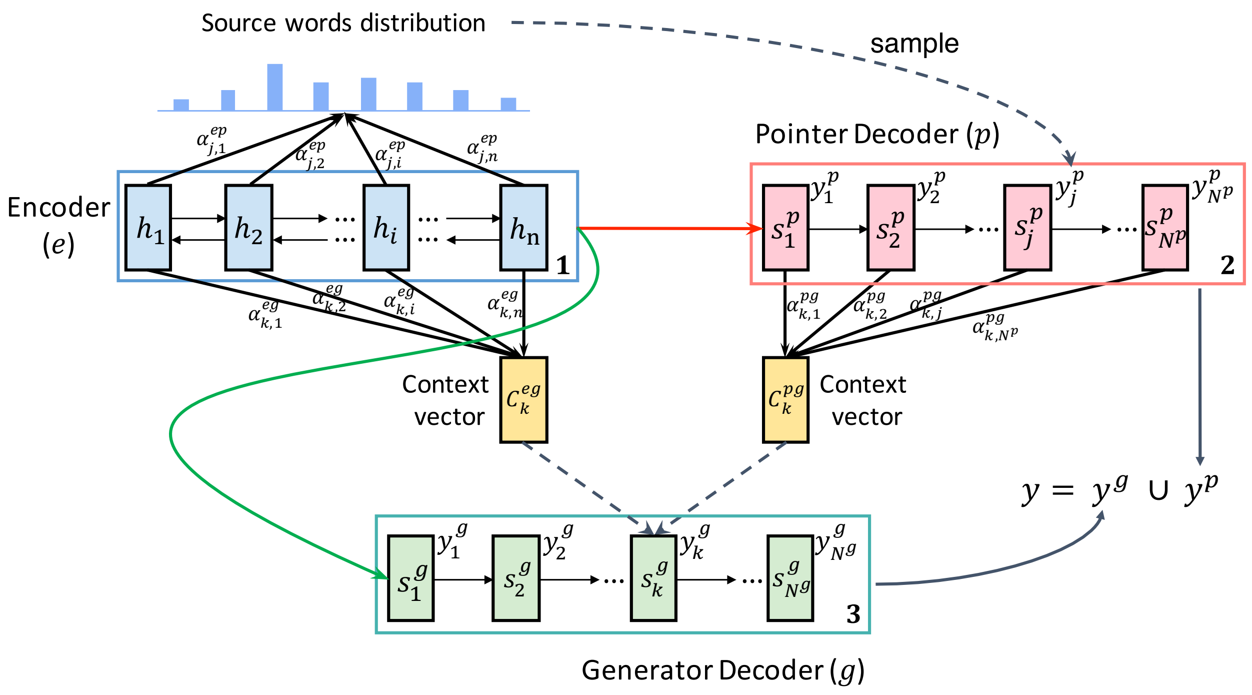


Figure 2: Overall architecture of our novel encoder-decoder model. The pointer decoder (p) first extracts words directly from the source text via copying mechanism. Then, the generator decoder (g) generates words from a fixed-size output vocabulary. Finally, we combine the outputs of the two decoders as the final result. For simplicity, we omit some links for computing the attention score α (see sections 3.3 and 3.4 for more details).

other 55% words can be extracted directly from $\mathbf{x}$. Therefore, during the training phase, we split $\mathbf{y}^{(i)}$ into two parts: $\mathbf{y}^{(i)^p}$ and $\mathbf{y}^{(i)^g}$. Here, $\mathbf{y}^{(i)^p}$ represents extractive keywords appearing in $\mathbf{x}^{(i)}$, which can be extracted by the pointer decoder (p), while $\mathbf{y}^{(i)^g}$ represents generative keywords not appearing in $\mathbf{x}^{(i)}$, which is generated by the generator decoder (g). When inferencing, we combine the outputs of the pointer decoder and generator decoder, and filter out the duplicates, generating the keyword query.

For the purpose of simplicity, except for a special statement, we will use $(\mathbf{x}, \mathbf{y}^p, \mathbf{y}^g)$ to represent each data sample $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)^g}, \mathbf{y}^{(i)^p})$ in the rest of this section.

3.2 The Neural Encoder

Just as with the general encoder-decoder model, our neural encoder uses a bidirectional RNN to encode a natural language query into a hidden states representation. In particular, we use the Long Short-Term Memory network (LSTM) [15] to replace the vanilla RNN. The LSTM introduces a gate mechanism and memory cell to avoid the gradient vanishing problem, which can provide better performance than the vanilla RNN in most cases.

Formally, let input sentence $\mathbf{x} = \{x_1, x_2, ..., x_n\}$. The forward LSTM reads $\mathbf{x}$ in a left-to-right order, resulting in a sequence of hidden states $[\overrightarrow{h}_1, \overrightarrow{h}_2, ..., \overrightarrow{h}_n]$. The backward LSTM reads $\mathbf{x}$ in a reverse order and generates the sequence of hidden states $[\overleftarrow{h}_1, \overleftarrow{h}_2, ..., \overleftarrow{h}_n]$. The forward and backward hidden states are concatenated to construct the final encoder hidden states $\boldsymbol{h} = [h_1, h_2, ..., h_n]$, where $h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$ contains information about the whole sentence and significantly focuses on the i-th word x_i and its surrounding words.

In our model, the hidden states $\boldsymbol{h}$ will provide dynamic context information of the source text both for the pointer decoder and the generator decoder by means of the attention mechanism.

3.3 The Neural Pointer Decoder

According to our statistics, about 55% keywords in the keyword query come from original natural language query $\mathbf{x}$. Hence, we first use a pointer decoder to extract extractive keywords directly from natural language query via copying mechanism [32].

The pointer decoder is an unidirectional LSTM with hidden states $\boldsymbol{s^p}$, and unfolds the encoder hidden states to extract word from source text $\mathbf{x}$ at each time step. In our model, the pointer decoder is initialized with the summarization of the entire source text $[\overrightarrow{h}_n, \overleftarrow{h}_1]$, which is defined as follow:

$$s_0^p = W_s^p[\overrightarrow{h}_n, \overleftarrow{h}_1] \tag{1}$$

where W_s^p is a trainable parameter matrix, which learns to map the concatenation of forward encoder final hidden state $\overrightarrow{h}_n$ and backward encoder final hidden state $\overleftarrow{h}_1$ to the semantic spaces of the pointer decoder.

In order to make the decoder dynamically focus on the important parts of the input, an attention mechanism is used. The attention distribution $\alpha_{j,i}^{ep}$ over the encoder hidden state h_i at time j is calculated as in Bahdanau et al. [2]:

$$\alpha_{j,i}^{ep} = \frac{exp(e_{j,i}^{ep})}{\sum_{i'}^{N} exp(e_{j,i'}^{ep})} \quad (2)$$

$$e_{j,i}^{ep} = v_{ep}^{T} \tanh(W_h^{ep} h_i + W_s^{ep} s_j^{p}) \quad (3)$$

where $v_{ep}^T, W_h^{ep}, W_s^{ep}$ are learnable parameters of the encoder-pointer decoder attention module, and s^p is the hidden states of the pointer decoder. We can regard the attention distribution as a probability distribution over the words in the source text, which tells the decoder which word should be extracted as the current output. For each source text $\mathbf{x} = \{x_1, x_2, ..., x_n\}$, we have a vocabulary χ that consists of all the words in $\mathbf{x}$. The pointer decoder can extract a word from χ as the current output by sampling from the probability distributions for χ. Specifically, we selected the word of maximum probability as the current output in our experiments. The probability of any word $y_j^p \in \chi$ is defined as follows:

$$P_c(y_j^p | \mathbf{y}_{<j}, \mathbf{x}) = \sum\nolimits_{i: x_i = y_j^p} \alpha_{j,i}^{ep} \quad (4)$$

In order for the pointer decoder can automatically stop the decoding process when inferring, we add an extra special token sos^p as the end flag at the end of the source text. The pointer decoder will stop the decoding process when sos^p is selected.

3.4 The Neural Generator Decoder

Although the pointer decoder could extract partial keywords directly from the source text, there are some generative keywords that do not appear in the source text. The generation of these keywords relies on an understanding of the content of the source text. Hence, we use a generator decoder with attention modules to generate keywords from a fixed-size output vocabulary. This vocabulary is composed of the most common query keywords.

The generator decoder is also an unidirectional LSTM, and its initial states is defined as follows:

$$s_0^g = W_s^g [\overrightarrow{h_n}, \overleftarrow{h_1}] \quad (5)$$

where W_s^g is a trainable parameter matrix, which learns to map the concatenation of forward encoder final hidden state $\overrightarrow{h_n}$ and backward encoder final hidden state $\overleftarrow{h_1}$ to the semantic spaces of the generator decoder.

As shown in Figure 2, to enable the decoder could focus on the important parts in the source text, we use the encoder - generator decoder attention module to dynamically capture the source-side context. At each decoding step k, the context vector c_k^{eg} is a weighted sum of the encoder hidden states h:

$$c_k^{eg} = \sum_{i=1}^{N} \alpha_{k,i}^{eg} h_i \quad (6)$$

$$\alpha_{k,i}^{eg} = \frac{exp(e_{k,i}^{eg})}{\sum_{i'}^{N} exp(e_{k,i'}^{eg})} \quad (7)$$

$$e_{k,i}^{eg} = v_{eg}^{T} \tanh(W_h^{eg} h_i + W_s^{eg} s_k^{g}) \quad (8)$$

where $v_{eg}^T, W_h^{eg}, W_s^{eg}$ are learnable parameters of the encoder-generator decoder attention module, and s^g is the hidden states of the generator decoder. Furthermore, the keywords extracted by the pointer decoder may also contribute to the generation process, so we use another pointer decoder - generator decoder attention module to dynamically capture the target-side context of the pointer decoder. The context vector c^{pg} is calculated as follows:

$$c_k^{pg} = \sum_{j=1}^{N^p} \alpha_{k,j}^{pg} s_j^p \quad (9)$$

$$\alpha_{k,j}^{pg} = \frac{exp(e_{k,j}^{pg})}{\sum_{j'}^{N^p} exp(e_{k,j'}^{pg})} \quad (10)$$

$$e_{k,j}^{pg} = v_{pg}^{T} \tanh(W_h^{pg} s_j^p + W_s^{pg} s_k^{g}) \quad (11)$$

where $v_{pg}^T, W_h^{pg}, W_s^{pg}$ are learnable parameters of the pointer decoder-generator decoder attention module; and s^p and s^g are the hidden states of the pointer decoder and generator decoder, respectively. It is worth noting that we directly use the hidden state sequence rather than the word sequence generated by the pointer decoder as Zhang et al [36]. In this manner, our model could better avoid negative effect of translation prediction errors.

Then, the encoder context vector c^{eg}, pointer decoder context vector c^{pg}, and current hidden state s_k^g are concatenated, and fed through a single feed-forward layer and a softmax layer to produce the probability distribution over the fixed-size output vocabulary :

$$P_g(y_k^g | \mathbf{y}_{<k}, \mathbf{x}) = softmax(W_o[s_k^g, c_k^{eg}, c_k^{pg}] + b_o) \quad (12)$$

where W_o, b_o are trainable parameters of single feed-forward layer.

3.5 Training and Inference

The most widely used method to train the encoder-decoder model is called the "teacher forcing" algorithm [33], which minimizes the sum of cross entropy loss of the ground-truth outputs. Given a training corpus $D = \{(\mathbf{x}, \mathbf{y}^p, \mathbf{y}^g)\}$, the objective function is defined as follow:

$$L(D) = \frac{-1}{|D|} \sum_{\mathbf{x}, \mathbf{y} \in D} \{\lambda \cdot logP(\mathbf{y}^p | \mathbf{x}) + (1 - \lambda) \cdot logP(\mathbf{y}^g | \mathbf{x})\} \quad (13)$$

where $\lambda \in [0, 1]$ is a hyper-parameter accounting for the preference in magnitude between the two loss term. The loss term $logP(\mathbf{y}^p|\mathbf{x})$ models the translation procedure of the pointer decoder, while $logP(\mathbf{y}^g|\mathbf{x})$ models the translation procedure of the generator decoder. These sub-objective functions are defined as follows:

$$logP(\mathbf{y}^p | \mathbf{x}) = \sum_{j=1}^{N^p} logP_c(y_j^p | \mathbf{y}_{<j}, \mathbf{x})$$

$$logP(\mathbf{y}^g | \mathbf{x}) = \sum_{k=1}^{N^g} logP_g(y_k^g | \mathbf{y}_{<k}, \mathbf{x}) \quad (14)$$

During inferring, we adopt a three-phase scheme to get the keyword query. First, we use the pointer decoder to sequentially extract a word from the source text as the keyword until the special token sos^p is selected. Then, the generator decoder generates a keyword from the fixed-size vocabulary until the special token sos^g

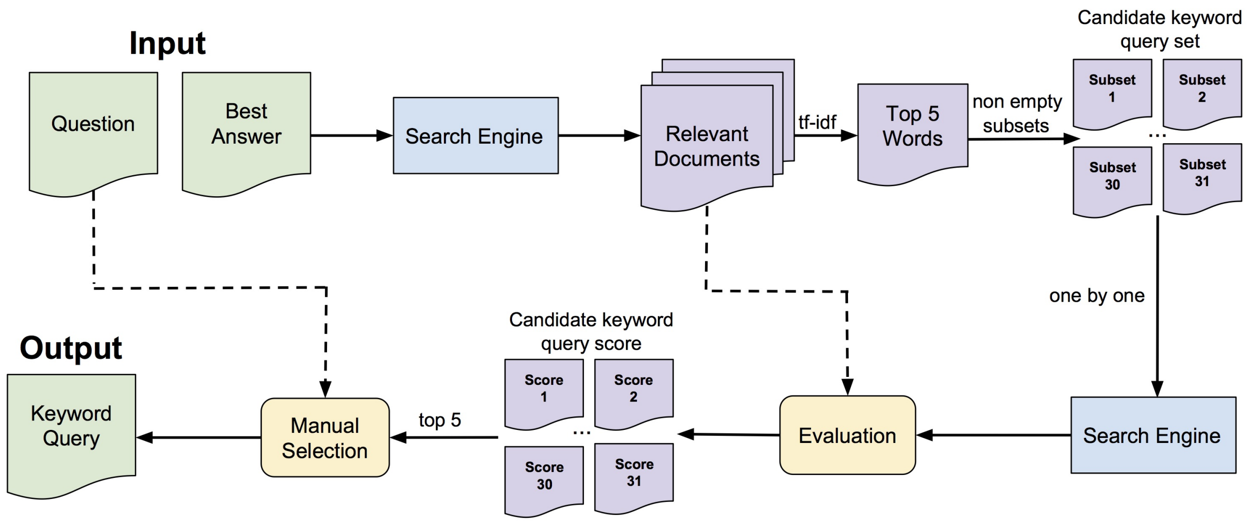


Figure 3: The semi-automatic process of data annotation. We first used the best answer to search relevant documents for each question. Then, all subsets of top 5 words of TF-IDF score in the relevant documents would be set as the candidate keyword query set. Next, we made an automatic evaluation for each candidate keyword query and calculated its score. Finally, we manually selected the best keyword query from the candidate set.

is selected. Finally, we combine the outputs of the pointer decoder and generator decoder and filter out the duplicates, generating the final keyword query.

4 DATA CONSTRUCTION

In this section, we will introduce the semi-automatic data construction process in details. Our constructed dataset is composed of a collection of triples (*natural language query* (*question*), *best answer*, *keyword query*). The natural language query corresponds to the question of "Baidu Knows". The best answer is the best one selected by the questioner from all answers, or it is the answer with the most votes. The keyword query is constructed by extracting some words from the relevant documents. In particular, the relevant documents are the documents that are similar to the best answer or contain the best answer. The key point of the dataset construction is how to define relevant documents and generate query keywords, and we will describe them in detail below.

Specially, there are two reasons that we selected "Baidu Knows" to construct the dataset. On the one hand, questions from the CQA website are pretty close to users' true natural language queries, and they are relatively easy to obtain. On the other hand, it is difficult for an academic community to obtain query logs from a commercial search engine. Hence, we can not follow the practice of constructing a dataset using query logs as [8, 17, 25, 30] .

4.1 Data Collection

We first randomly crawled about 1 million questions and each question's corresponding answers from "Baidu Knows". Because there could be errors or irrelevant answers for each question, we only reserved the best answer for each question to reduce noisy data. Meanwhile, we deleted those questions that were repeated or had no best answer. After this operation, approximately 300 thousand question-best answer pairs remained.

In order to obtain reasonable and high-quality question - best answer pairs, we further imposed the following restrictions: a) the question had no less than 15 characters, b) the best answer had no less than 300 characters, and c) the votes for the best answer were no less than 10. Approximately 85K question-best answer pairs satisfied these restrictions, and we selected them as the question-best answer pairs for data annotation.

4.2 Data Annotation

Figure 3 illustrates the semi-automatic process for data annotation. Given a question-best answer pair, the purpose of data annotation was generating its corresponding keyword query. We first submitted the best answer to each question into the Baidu search engine and calculated the relevance between best answer and each retrieved document of first three pages. Here, we used *Edit Distance Ratio* (*EDR*) [28] to represent the relevance. Concretely, the *EDR* is defined as:

$$EDR\,(\text{a,b}) = 1 - \frac{ED\,(\text{a,b})}{|\text{a}| + |\text{b}|} \tag{15}$$

where |a| and |b| are respectively the length of string a and b. The *ED* represents *Edit Distance*: the minimum number of operations required to transform one string into another string. The *EDR* ranges from 0 to 1. In particular, we randomly selected 100 best answer and their retrieved documents, and make a manual evaluation on whether the retrieved documents were similar to the best answer or not. According to evaluation results, we found that two documents were similar in 91% case if their *EDR* is no less than 0.6. Hence, we selected those retrieval documents whose relevance was no less than 0.6 as **relevant documents**.

Table 1: Statistical information of the data construction

Description	Value
Original file size of crawled data	1.68 GB
Original crawled questions	98.6K
Question-best answer pair	85362
Manual selection question	21483
Question-keyword query pair	11296
Average number of relevant documents	8.3
Average keywords number of keyword query	3.05

After obtaining the relevant documents, we used TF-IDF to score each word in the relevant documents and selected the **top 5 words**. Then, all possible non empty subsets of the top 5 words were used to construct the **candidate keyword query set**, which had a size of 31. Next, we made an automatic **evaluation** for each candidate keyword query using the "Baidu" search engine. Concretely, we used the "Baidu" to retrieve each candidate keyword query, and use the number of relevant documents in the top 10 retrieved documents as the **candidate keyword query score**. Notably, we filtered out those questions whose candidate keyword queries scores were all small than 2. About 21K questions remained.

Then, we invited 5 people to make a **manual selection**, in which the best keyword query would be selected from the top 5 score candidate keyword query according to the score and the original question. Meanwhile, we filtered out unreasonable question-keyword query pairs. The process takes about 210 hours in total. Finally, about 11 K questions with their corresponding keyword queries and best answers, remained, and we used them to train and evaluate our model.

Table 1 gives the related data statistics information in the data collection and data annotation process.

5 EXPERIMENT

5.1 Dataset and Setup

To train and evaluate the proposed method, we split our constructed dataset into a training set and a testing set with a ratio of 8:2, and randomly selected 10% of the training set as the development set. Because the dataset we constructed was derived from a Chinese CQA site, we first used *jieba* [3] to conduct word segmentation (divide the sentences into words) for the dataset. In addition, we used the largest Chinese search engine "Baidu" as the test search engine, because it has better retrieval performance than other search engines (e.g. Google or Bing) for Chinese queries.

In the experiments, the word embeddings were pre-trained on the Chinese Wikipedia corpus using the word2vec [23] toolkit and fine-tuned in the training process. The dimension of the word embeddings was 60. We used a single layer's bidirectional LSTM as the encoder, and another two single layer's LSTM as the pointer decoder and generator decoder, respectively. The dimensions of the LSTMs both for the encoder and decoders were all set to 128. The word vocabulary sizes for the natural language queries and keyword queries respectively were 20130 and 9613, respectively. In particular, the encoder and decoders shared the same word embeddings. During training, the models were optimized using Adam [18] optimization algorithm with a batch size of 64, an initial learning rate of 0.001, a λ of 0.5, and a gradient clipping of 5. Dropout [31] regularization has proved to be an effective method for reducing the overfitting in neural networks with millions of parameters. In this work, we also used it to improve the regularization of the hidden layer, and the dropout rate was set to 0.4. This training process was stopped when the loss of the development dataset stopped dropping for several epochs. These configurations were also used in the other models described in the following paragraphs.

It should be noted that, according to our statistics, only 55% of the keywords in the keyword queries appeared in the original natural language queries, while the other 45% keywords did not appear in the natural language queries. In general, the generation of these 45% keywords was more difficult, which required an understanding of the meaning of natural language queries.

5.2 Evaluation

First, we used the precision (P), recall (R), and F1-score (F1) to evaluate the performance when translating natural language queries into keyword queries. The precision was calculated based on the proportion of keywords truly predicted among all the keywords predicted by the model. The recall was calculated based on the proportion of truly predicted among the golden standard keywords.

Furthermore, we also used two common metrics (Hits@K and Precision@K) to evaluate the retrieval performance for original language queries, as well as translated keyword queries using different methods. In particular, because original natural language queries all come from "Baidu Knows", for all methods, we filtered out all retrieved documents that from "Baidu Knows" in order to make a fair comparison.

(1) *Hits*@K: Hits at top-K retrieved documents:

$$\text{Hits@K} = \frac{1}{|Q|} \sum_{q \in Q} \text{h}_\text{K}(q) \tag{16}$$

where Q is the queries set and $|Q|$ is the size of the queries set. h_K(q) represents whether top-K retrieved documents contains any relevant document of q:

$$\text{h}_\text{K}(q) = \begin{cases} 1 & \text{if any relevant document of } q \text{ is contained} \\ 0 & \text{otherwise} \end{cases}$$

Specifically, those retrieved documents whose relevance (see Eq. 15) with the best answer was no less than 0.6 would be regarded as the relevant documents in our work.

(2) *Precision@K*: Precision of the top-K retrieved documents:

$$\text{P@K} = \frac{|D_K \cap D^*|}{|D_K|} \tag{17}$$

where D_K are top-K retrieved documents, and D^* are the relevant documents. Here, D^* are those retrieved documents whose relevance (see Eq. 15) with the best answer was no less than 0.6. P@K represents the proportion of relevant documents among the retrieved documents.

[3]Jieba is a popular open source project for Chinese word segmentation, which is available at https://github.com/fxsjy/jieba.

Table 2: **Performance of the different methods on the testing set.**

#	Method	Retrieval performance					Translation performance		
		Hits@5	Hits@10	P@3	P@5	P@10	Precision	Recall	F1-Score
1	Raw Query	0.259	0.293	0.118	0.083	0.068	0.112	0.552	0.186
2	Vanilla Seq2seq:	0.191	0.217	0.079	0.057	0.047	0.250	0.222	0.235
3	Attentional Seq2seq	0.222	0.251	0.096	0.069	0.057	0.313	0.261	0.284
4	Sequence Labeling	0.314	0.355	0.146	0.109	0.093	0.589	0.387	0.467
5	Pointer Network	0.283	0.312	0.141	0.106	0.085	0.596	0.339	0.439
6	ATS2S + SL	0.298	0.342	0.132	0.100	0.082	0.387	0.521	0.444
7	ATS2S + PN	0.287	0.320	0.133	0.099	0.080	0.384	0.509	0.438
8	Joint ATS2S + SL	0.292	0.329	0.140	0.107	0.084	0.440	0.465	0.452
9	Our model	**0.387**	**0.441**	**0.164**	**0.124**	**0.100**	0.535	0.494	**0.514**

5.3 Methods for Comparison

For comparison with the proposed method, we evaluated the following methods on the constructed corpus:

- **Raw Query:** The original natural language query was directly given to the search engine without any modification, which could be regarded as a baseline level for retrieval performance.
- **Vanilla Seq2seq:** This was the general encoder-decoder model [3] without the attention mechanism. The encoder was a single layer's bidirectional LSTM, and the decoder was a single layer's unidirectional LSTM.
- **Attentional Seq2seq:** This was the general encoder-decoder model with the attention mechanism [20], which is already widely used in several NLP tasks.
- **Sequence Labeling:** This method is an extractive method, and it can only extract words from the original queries as the keywords of keyword queries. Concretely, we regarded the keywords extraction as a word-based sequence labeling task and adopted the Bi-LSTM model.
- **Pointer Network:** This method [32] is also an extractive method and it can be regarded as a variant of our proposed model without the generator decoder module.
- **ATS2S + SL:** This method directly combined the results of the attentional seq2seq and sequence labeling models. Then, it filtered out duplicated words to construct the keyword queries.
- **ATS2S + PN:** This method directly combined the results of the attentional seq2seq and pointer network models. Then it filtered out duplicated words to construct the keyword queries.
- **Joint ATS2S + SL:** The attentional seq2seq model and sequence labeling model share the same Bi-LSTM encoder. The sequence labeling model used the Bi-LSTM encoder to extract keywords. Meanwhile, the attentional seq2seq model generated keywords by its decoder. The two models were trained jointly together.

5.4 Results and Discussion

Table 2 lists the performances of the different methods on our constructed dataset. We regarded the raw query as a baseline level, which could give us an indication of whether these methods were effective for translating natural language queries. Its precision was approximately 0.112, which reflected the proportion of redundant words in the original queries. The recall was approximately 0.552, which reflected the proportion of keywords that came from the original queries. From the table, we can observe that our proposed model is significantly better than the other methods because it obtains the best results on these important metrics.

Vanilla Seq2seq and Attentional Seq2seq are both generative methods, which generate keywords directly from a fix-sized output vocabulary. In particular, Song et al. [30] proposed using the Attentional Seq2seq model to translate natural language queries into keyword queries. However, observing the results of their model, we see that the generative method is not effective in our task. We supposed that this was because there were more low frequency keywords and out-of-output-vocabulary keywords such as named entities in our constructed dataset. However, the Attentional Seq2seq model could not effectively handle these keywords. Sequence Labeling and Pointer Network were both extractive methods, which extracted words directly from the natural language queries to construct the keyword queries. However, these extractive models could not generate keywords that did not appear in the original queries. Because of the lexical chasm problem, about half of the keywords did not appear in the original queries. Therefore, these extractive methods could not fit the task well. It is notable that the extractive methods performed better than the generative methods for the task, which also indicated the difficulty of generating generative keywords directly from the output vocabulary.

Considering the results of ATS2S+SL and ATS2S+PN, we can observe that directly combining the results of generative methods and extractive methods did not produce better performance, and it was even poorer than the single extractive methods. We supposed that this direct combination might have generated more redundant words, which hurt the performance. The relatively higher recall and low precision could prove this supposition. In addition, we

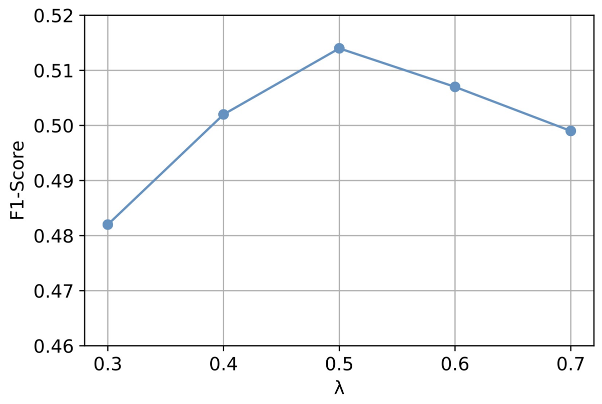


Figure 4: F1-score on the development set using different λ

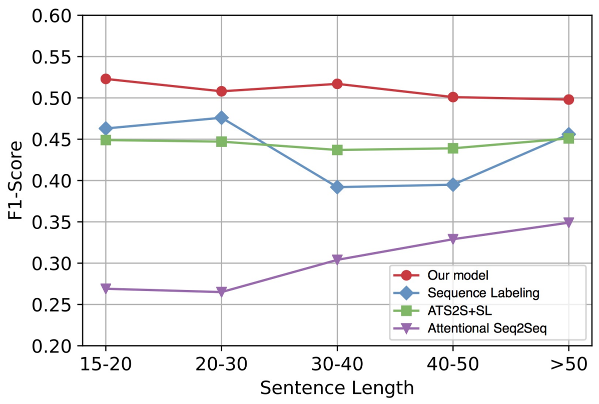


Figure 5: F1-score of the natural language queries at different lengths.

Table 3: The recall performance of different methods on the extractive keywords, generative keywords and total keywords.

# Method	Extractive Keywords	Generative Keywords	Total Keywords
Attentional Seq2seq	0.226	0.326	0.261
Sequence Labeling	0.656	0.000	0.387
Our Model	0.630	0.301	**0.494**

also attempted to jointly train the Attentional Seq2seq model and Sequence Labeling model in the Joint ATS2S + SL method. However, this did not work well. We suppose that this was because the encoder had difficulty simultaneously bear the source text information and extracting keyword information.

It should be noted that our model is equivalent to the Attentional Seq2seq model when the pointer decoder module is removed. Meanwhile, our model will transform into the pointer network model when the generator decoder module is removed. Our proposed model combines the Attentional Seq2seq and Pointer Network Models. From the results listed in Table 2, we can observe that our approach achieves a relative improvement of 17.1% for F1-score, 41.8% for Hits@10, 16.3% for P@3, and 17.0% for P@5 over the Pointer Network model. Meanwhile, our method also provides improvements of 81.0% for F1-score, 75.7% for Hits@10, 70.8% for P@3, and 79.7% for P@5 over the Attentional Seq2seq model.

5.5 Analysis

As stated, one important highlight of our work is that our model can not only extract extractive keywords that appear in the original natural language queries, but also generate those generative keywords that do not appear in the original natural language queries. Table 3 lists the recall performances of different methods on the extractive keywords, generative keywords and total keywords, based on a comparison of our proposed model with the Sequence Labeling model (extractive method) and Attentional Seq2seq model (generative method). From the result, we can see that the Attentional Seq2seq model was a little better at generating generative keywords, but it could not handle extractive keywords well. Meanwhile, the Sequence Labeling model could better extract extractive keywords, but it could not identify generative keywords. However, both the extractive keywords and generative keywords are important for retrieval. Our model could simultaneously handle extractive keywords and generative keywords well and achieved much better recall performances for total keywords than the other methods. Therefore, our model had the best performance in processing natural language queries.

Then, we investigated the impact of the hyper-parameter λ (see Eq. 13) on the development set. For this purpose, we gradually varied λ from 0.3 to 0.7 with an increment of 0.1 in each step. To simplify the experiments, we only adopted the F1-score as the evaluation criterion. As shown in Figure 4, we found that our model achieved the best performance when $\lambda = 0.5$. We believed that this was because the pointer decoder and generator decoder were equal in importance during the training process. Therefore, we set $\lambda = 0.5$ in the experiment.

We also investigated the performances of different methods at different lengths of the natural language queries. To simplify the experiments, we only adopted the F1-score as the evaluation criterion. We divided our testing sets into different groups and then compared the performances for each group. In particular, the maximum character length of the queries was 79, while the minimum character length of the queries was 15. Figure 5 illustrates the F1-score on these groups of testing sets. We observed that our model achieved the best performance in all the groups, although the performances of most systems slightly dropped with an increase in the length of the source sentences. These results clearly demonstrated once again the effectiveness of our model. In particular, we found that the Attentional Seq2seq model showed a converse trend, with a better performance when processing long queries than short queries. We think this is because that long sentence is more likely to be understood by the Attentional Seq2seq model.

6 CONCLUSION

In this work, we studied the problem of translating natural language queries into keyword queries, in order to alleviate the lexical chasm between the natural language queries and relevant documents.The task was regarded as a translation problem, in which the original natural language queries language were treated as one language, and the corresponding reformulated queries were treated as another language. We proposed a novel encoder-decoder model with two different decoders, respectively. This model could not only extract important keywords directly from the original natural language queries, but could also generate indispensable retrieval keywords that did not appear in the original queries. Because there is lack of large-scale publicly available dataset of true natural language queries, we also constructed a large-scale natural language query dataset. Comparing to the state-of-the-art deep neural networks, our approach achieves a better performance in processing natural language queries.

7 ACKNOWLEDGMENTS

The authors wish to thank the anonymous reviewers for their helpful comments. This work was partially funded by China National Key R&D Program (No. 2017YFB1002104), National Natural Science Foundation of China (No. 61532011, 61751201, 61473092, and 61472088), and STCSM (No.16JC1420401,17JC1420200).

REFERENCES

[1] Ioannis Antonellis, Hector Garcia Molina, and Chi Chao Chang. 2008. Simrank++: query rewriting through link analysis of the click graph. *Proceedings of the VLDB Endowment* 1, 1 (2008), 408–421.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *EMNLP*.

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

[4] Sumit Chopra, Michael Auli, and Alexander M. Rush. 2016. Abstractive Sentence Summarization with Attentive Recurrent Neural Networks. In *HLT-NAACL*.

[5] Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. 2002. Probabilistic query expansion using query logs. In *Proceedings of the 11th international conference on World Wide Web*. ACM, 325–332.

[6] Laura Dietz and Ben Gamari. 2017. TREC CAR: A Data Set for Complex Answer Retrieval. Version 1.4.(2017).

[7] Bruno M Fonseca, Paulo Golgher, Bruno Pôssas, Berthier Ribeiro-Neto, and Nivio Ziviani. 2005. Concept-based interactive query expansion. In *Proceedings of the 14th ACM international conference on Information and knowledge management*. ACM, 696–703.

[8] Jianfeng Gao, Xiaodong He, Shasha Xie, and Alnur Ali. 2012. Learning lexicon models from search logs for query expansion. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 666–676.

[9] Jianfeng Gao and Jian-Yun Nie. 2012. Towards concept-based translation models using search logs for query expansion. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 1.

[10] Alex Graves and Navdeep Jaitly. 2014. Towards End-To-End Speech Recognition with Recurrent Neural Networks. In *ICML*.

[11] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. *CoRR* abs/1603.06393 (2016).

[12] Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. *arXiv preprint arXiv:1603.08148* (2016).

[13] Shizhu He, Cao Liu, Kang Liu, and Jun Zhao. 2017. Generating Natural Answers by Incorporating Copying and Retrieving Mechanisms in Sequence-to-Sequence Learning. In *ACL*.

[14] Yunlong He, Jiliang Tang, Hua Ouyang, Changsung Kang, Dawei Yin, and Yi Chang. 2016. Learning to Rewrite Queries. In *CIKM*.

[15] Sepp Hochreiter and year=1997 volume=9 8 pages=1735-80 Jürgen Schmidhuber, journal=Neural computation. [n. d.]. Long Short-Term Memory. ([n. d.]).

[16] Samuel Huston and W Bruce Croft. 2010. Evaluating verbose query processing techniques. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 291–298.

[17] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. 2006. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web*. ACM, 387–396.

[18] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[19] Giridhar Kumaran and Vitor R Carvalho. 2009. Reducing long queries using query quality predictors. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 564–571.

[20] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *EMNLP*.

[21] K Tamsin Maxwell and W Bruce Croft. 2013. Compact query term selection using topically related text. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 583–592.

[22] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. *CoRR* abs/1609.07843 (2016).

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[24] Ramesh Nallapati, Bowen Zhou, Cícero Nogueira dos Santos, and year=2016 ÃĞaglar GülÃğehre and Bing Xiang, booktitle=CoNLL. [n. d.]. Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond.

[25] Stefan Riezler and Yi Liu. 2010. Query rewriting using monolingual statistical machine translation. *Computational Linguistics* 36, 3 (2010), 569–582.

[26] Stefan Riezler, Yi Liu, and Alexander Vasserman. 2008. Translating queries into snippets for improved query expansion. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*. Association for Computational Linguistics, 737–744.

[27] Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A Neural Attention Model for Abstractive Sentence Summarization. In *EMNLP*.

[28] Sandip Sarkar, Dipankar Das, Partha Pakray, and Alexander Gelbukh. 2016. JUNITMZ at SemEval-2016 Task 1: Identifying Semantic Similarity Using Levenshtein Ratio. In *International Workshop on Semantic Evaluation*.

[29] Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In *ACL*.

[30] Hyun-Je Song, A Kim, Seong-Bae Park, et al. 2017. Translation of Natural Language Query Into Keyword Query Using a RNN Encoder-Decoder. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 965–968.

[31] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15 (2014), 1929–1958.

[32] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *NIPS*.

[33] Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1, 2 (1989), 270–280.

[34] Jinxi Xu and W Bruce Croft. 1996. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 4–11.

[35] Wei Vivian Zhang and Rosie Jones. 2007. Comparing click logs and editorial labels for training query rewriting. In *WWW 2007 Workshop on Query Log Analysis: Social And Technological Challenges*.

[36] Xiangwen Zhang, Jinsong Su, Yue Qin, Yang Liu, Rongrong Ji, and Hongji Wang. 2018. Asynchronous Bidirectional Decoding for Neural Machine Translation. In *AAAI*.