Generating Synthetic Data for Neural Keyword-to-Question Models

Heng Ding
Wuhan University
Wuhan, China
hengding@whu.edu.cn

Krisztian Balog
University of Stavanger
Stavanger, Norway
krisztian.balog@uis.no

ABSTRACT

Search typically relies on keyword queries, but these are often semantically ambiguous. We propose to overcome this by offering users natural language questions, based on their keyword queries, to disambiguate their intent. This keyword-to-question task may be addressed using neural machine translation techniques. Neural translation models, however, require massive amounts of training data (keyword-question pairs), which is unavailable for this task. The main idea of this paper is to generate large amounts of synthetic training data from a small seed set of hand-labeled keyword-question pairs. Since natural language questions are available in large quantities, we develop models to automatically generate the corresponding keyword queries. Further, we introduce various filtering mechanisms to ensure that synthetic training data is of high quality. We demonstrate the feasibility of our approach using both automatic and manual evaluation.

CCS CONCEPTS

• **Information systems** → **Query intent**;

KEYWORDS

Keyword-to-question, synthetic data generation, neural machine translation

ACM Reference Format:
Heng Ding and Krisztian Balog. 2018. Generating Synthetic Data for Neural Keyword-to-Question Models. In *ICTIR '18: 2018 ACM SIGIR International Conference on the Theory of Information Retrieval, Sept. 14–17, 2018, Tianjin, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3234944.3234964

1 INTRODUCTION

Most search queries are motivated by some underlying question [13]. Today's users are accustomed to expressing the questions they have in mind using keyword queries [20]. Keyword queries, however, can be notoriously ambiguous and may be interpreted in multiple ways. For example, given the keyword query *"10th president India,"* the question perhaps most users would want to ask is *"Who was the 10th President of India?"*. Nevertheless, some users may be interested in a particular aspect of the query topic, like *"In which year did the 10th President of India leave office?"* or *"What do people say about the 10th President of India?"*. By determining the underlying question, we can obtain a more accurate representation of the user's information need. This, in turn, can lead to improved retrieval performance and a better overall search experience. We envisage a search interface that allows users to refine their queries with automatically generated natural language questions; see Fig. 1. Our goal is to automatically generate a natural language question that most likely represents the user's underlying information need. It is seen as a feedback mechanism that can more naturally engage users into explicitly clarifying their information needs. How those natural language questions are actually utilized in a retrieval system (e.g., via query expansion [13]) is beyond the scope of this study.

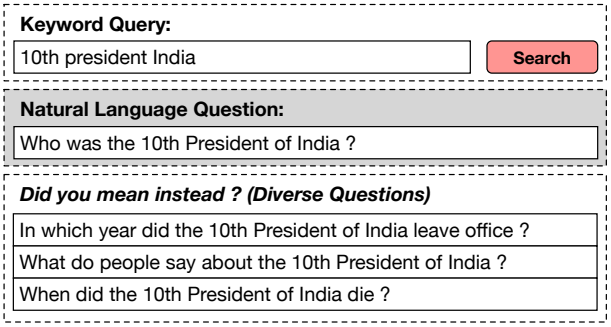

Figure 1: Translating a keyword query to natural language question(s). Our focus is on the shaded area: generating the most common question for a keyword query. The bottom part, generating diverse questions, is left for future work.

In this paper, we address the *keyword-to-question* (K2Q) task: generating a natural language question from a keyword query. K2Q has generated considerable attention recently, see, e.g., [7, 13, 20, 21]. All these systems follow a template-based approach, and are evaluated in terms of relevance, diversity, and grammatical correctness. While some differences exist among these systems, all consist of three main steps. First, they extract question templates from millions of keyword-question pairs by substituting keyword terms in questions with slots, and storing keyword-template pairs in a database $D_{\langle q,t\rangle}$. Second, given a new keyword query q', they search similar keyword queries from $D_{\langle q,t\rangle}$, collect templates related to those similar queries, and instantiate those templates with q' for generating candidate questions. Finally, a parameterized ranking model is used to calculate the probabilities of those candidate questions being generated by the query q', and to rank all candidate questions. However, these template-based methods are inherently limited in their ability

ICTIR'18, September 14–17, 2018, Tianjin, China

ACM ISBN 978-1-4503-5656-5/18/09…$15.00
https://doi.org/10.1145/3234944.3234964

to generalize to previously unseen queries. Instead, we propose to address the K2Q task using state-of-the-art neural machine translation (sequence-to-sequence) approaches. One challenge we face is that training such neural models requires massive amounts of training data (i.e., hand-labeled keyword-question pairs). While such training data could be mined from query and click logs, there are two main issues. First, such click data is not always available (e.g., in a cold start scenario). Second, it is limited to keyword-question pairs that have received sufficiently many clicks; long-tail queries or newly posted questions will not have that. The above considerations give rise to the main research objective of the present work: *How can we generate synthetic data for training a neural machine translation approach for the K2Q task?*

The idea of generating synthetic data for training deep neural network has already been successfully applied for some computer vision tasks [11, 18]. In information retrieval, prior work has studied the creation of pseudo test collections, i.e., automatically generating query-document pairs, for training and evaluating retrieval algorithms [1, 4]. Inspired by those studies, we propose an approach that automatically generates large amounts of simulated keyword-question pairs from a small set of hand-labeled keyword question pairs, and then learns a neural keyword-to-question model with such synthetic training data. The main technical contributions of this work are the following:

(1) We present a novel approach for generating synthetic training data from a seed set of hand-labeled keyword-question pairs, and subsequently use this data for learning neural machine translation models to solve the K2Q task (Sect. 2).
(2) We introduce several generative models for producing synthetic keyword queries from natural language questions (Sect. 3.1).
(3) We develop two filtering mechanisms, which are essential for ensuring that the synthetic training data we feed into the neural network is of high-quality (Sect. 3.2).
(4) We evaluate our synthetic data generation approach on the end-to-end K2Q task using both automatic and manual evaluation (Sect. 6).

2 OVERVIEW

The overall goal in this paper is to tackle the keyword-to-question (K2Q) problem using neural networks. I.e., the task is to translate a keyword query (referred to as *keyword* for short) to a natural language question (*question* for short). To be able to use neural networks for this task, massive amounts of training data are needed. The main idea of our paper is to use a small seed set of hand-labeled training data to generate large amounts of synthetic training data. Specifically, the *seed training data*, $\mathcal{T}_0$, consists of keyword-question pairs, $(k, q) \in \mathcal{T}_0$. This, along with a large *question corpus*, Q, is utilized to generate *synthetic training data*, $\mathcal{T}$, which also consists of keyword-question pairs, $(k, q) \in \mathcal{T}$. The neural machine translation models will then be trained using $\mathcal{T}$. The overview of our framework is shown in Fig. 2. It entails three main steps, which we shall detail below.

First, we train a keyword query generation model (KQGM), θ, using keyword-question pairs from the seed training data. We aim to simulate real users' querying behavior: given a natural language question, generate a keyword query that a user would likely issue

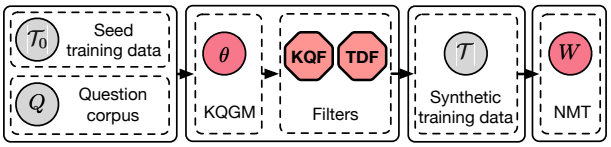

Figure 2: The overview of our approach.

when seeking an answer to that question. We explore various generative models; these have only a few free parameters, which can be easily learned from the seed training data $\mathcal{T}_0$.

Second, we utilize a large question corpus Q, collected from community question answering forums, and employ the keyword query generation model θ to generate (a large set of) simulated keyword-question pairs. These will constitute our synthetic training data $\mathcal{T}$. However, since not all the automatically generated keyword-question pairs are of high quality, we employ a keyword query filter (KQF) and a training data filter (TDF). These filters are pivotal elements in our approach; we shall detail them in Sect. 3.2.

Finally, we train a neural machine translation (NMT) model for the K2Q task by feeding it with the synthetic training data $\mathcal{T}$. We consider three neural networks: basic encoder-decoder NMT [19], NMT with attention mechanism [3], and NMT with copying mechanism [10].

3 SYNTHETIC DATA GENERATION

This section details the our synthetic training data generation method, which is the most important contribution of this paper. The process takes as input (i) a small seed training data set, consisting of hand-labeled keyword-query pairs, and (ii) a large set of natural language questions. The output is a large set of automatically generated keyword-question pairs, with high enough quality to train robust neural models. Our approach consists of two main components: a keyword query generation model (Sect. 3.1) and filtering mechanisms (Sect. 3.2).

3.1 Keyword Query Generation Model

Prior work has seen successful attempts at generating synthetic queries for web and microblog known-item search, both for evaluation and for training purposes [1, 4]. The overall idea is to construct a generative model that can produce a query, similar to a real query that a user would issue, for finding a particular item. We take the algorithm proposed by Azzopardi et al. [2] as our starting point (§3.1.1) and extend it at several points to fit our problem setting: (i) we impose a number of restrictions as well as introduce new elements to the generative process (§3.1.2), (ii) we propose a paraphrase-based variation that considers multiple ways of formulating the same question (§3.1.3), and (iii) we add phrase support, so as not to break up meaningful word sequences (§3.1.4).

3.1.1 Baseline. In known-item search it is assumed that the user wants to find a particular item (document, question, tweet, etc.) that she has seen before in the corpus. Therefore, the user constructs a keyword query by recalling terms that would help her identify this item. In automatic query construction this user behavior is simulated using generative models. Formally, let us assume that the user seeks to find (recall) the natural language question q. The query

length s is selected with probability $P(s)$. Then, a keyword query $k = (t_1, \ldots, t_s)$ is constructed by sampling s terms from $P(t_i|\theta_q)$, which is the model of q. The prior probability distribution $P(s)$ can be easily estimated by considering query lengths in a representative sample (e.g., a query log). The quality of the synthetic queries crucially depends on the distribution $P(t_i|\theta_q)$, as it determines which terms will be sampled. Azzopardi et al. [2] define $P(t_i|\theta_q)$ using the standard language modeling approach:

$$P(t_i|\theta_q) = (1-\lambda)P(t_i|q) + \lambda P(t_i) \ . \tag{1}$$

Accordingly, term generation is a mixture between sampling a term from the given item with probability $P(t_i|q)$, and from the corpus with probability $P(t_i)$, where the influence of the collection model is controlled by the smoothing parameter λ. The latter likelihood is calculated using:

$$P(t_i) = \frac{n(t_i)}{\sum_{t_j \in V} n(t_j)} \ , \tag{2}$$

where $n(t_i)$ denotes the collection term frequency of term t_i, and V is the vocabulary of terms in the corpus.

To simulate different types of user querying behavior, three plausible term selection strategies have been proposed to estimate $P(t_i|q)$: (i) popular selection, (ii) discriminative selection, and (iii) their combination [1, 2].

(i) *Popular*: Assuming that more frequent terms are more likely to be used as query terms, $P(t_i|q)$ is calculated by Eq. (3), where $n(t_i, q)$ is the number of occurrences of t_i in q.

$$P(t_i|q) = \frac{n(t_i, q)}{\sum_{t_j \in q} n(t_j, q)} \tag{3}$$

(ii) *Discriminative*: Assuming that the user may select query terms that can better discriminate the item she is looking for from other items in the corpus, $P(t_i|q)$ is calculated using Eq. (4), where $b(t_i, q)$ is a binary indicator function that is 1 if t_i occurs in q and 0 otherwise. $P(t_i)$ is the same as before, cf. Eq. (2).

$$P(t_i|q) = \frac{b(t_i, q)}{P(t_i) \sum_{t_j \in q} \frac{b(t_j, q)}{P(t_j)}} \tag{4}$$

(iii) *Combination*: Combining the popular and discriminative strategies into a single model, $P(t_i|q)$ is calculated by Eq. (5), where $df(t_i)$ is the document (here: question) frequency of term t_i and N is the total number of items in the corpus.

$$P(t_i|q) = \frac{n(t_i, q) \log \frac{N}{df(t_i)}}{\sum_{t_j \in q} (n(t_j, q) \log \frac{N}{df(t_j)})} \tag{5}$$

3.1.2 Our Keyword Generation Algorithm. Note that the original algorithm in [2] has been developed for known-item (document) search. We need to modify and extend it at several points to be able to use it for the K2Q task we are addressing.

For known-item search, an item q is selected randomly from the corpus, and then a keyword query is generated from that item. The process is repeated as many times as the number of queries to be created. In our problem scenario the items are natural language questions, where each of them needs to be paired with a keyword query. That is, we do not sample items, but we generate a query for each item in the corpus. This is the first modification we make to the algorithm (line 3 in Algorithm 1).

Algorithm 1: Synthetic keyword-question generation

Data: Q, a set of known questions
Result: $\langle \mathcal{K}, \mathcal{Q} \rangle$, a set of synthetic keyword-question pairs

```
1  begin
2      ⟨K, Q⟩ ← ∅;
3      for q ∈ Q do
4          k ← [];
5          s ← sampleQueryLength(P(s));
6          for j in [1, s], s < |q| do
7              t_i ← sampleTerm(P(t_i|θ_q));
8              k' ← append(k, t_i);
9              P(t_i|θ_q) ← 0;
10         end
11         ⟨K, Q⟩ ← ⟨K, Q⟩ ∪ {⟨k, q⟩};
12     end
13 end
```

The second change concerns the length of keyword queries. In [2], the length of the query is drawn from a Poisson distribution, with the mean set according to the average length in a set of human-generated queries. For us, the length of the keyword query also depends on the corresponding natural language question. Given a question with length $|q|$, it is reasonable to assume that users will always prefer to issue a keyword query that is shorter than $|q|$. Thus, we include this additional constraint and sample a query length with $P(s)$, where $s < |q|$ (line 5 in Algorithm 1).

Third, keyword queries typically do not contain question words, such as "how," "what," "where," "who," "why," "when," etc. Thus, we do not sample question words in our generation process.

Fourth, our algorithm does not only sample terms but also samples phrases for generating synthetic queries. Thus, we avoid breaking up word sequences that function together as a meaningful unit. It means that t_i could be either a term or a phrase in the generative process (line 7 in Algorithm 1). We describe our phrase detection mechanism in §3.1.4.

Fifth, according to our statistics on a sample of queries,[1] only 3.9% of all keyword queries include the same term more than once, suggesting that queries with repeated terms are atypical. Thus, we find it reasonable to avoid sampling the same term more than once in our keyword query generation process (line 9 in Algorithm 1).

3.1.3 Paraphrase-Based Querying Model. Users may use different words to express the same meaning. This should be taken into consideration in the keyword query generation process. Imagine the following case, where a particular user has seen the question *"Who is the author of the pooh?"* in a community question answering forum (e.g., Yahoo! Answers or Quora), then, after several days, she tries to recall the search terms to find an answer to this question. If she still remembers the exact words from the question, she may issue *"the pooh author"* as a query. Otherwise, she may recollect a paraphrase of the question, like *"Who is winnie the pooh's creator?"* and, based on that, formulates the keyword query *"winnie the pooh creator."* Furthermore, different users may recall different

[1] The Yahoo! L16 Webscope Dataset, which contains many real keyword queries from users of Yahoo Answers.

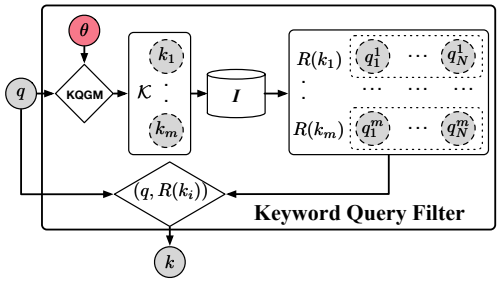

Figure 3: The architecture of our keyword query filter (KQF).

paraphrases during their querying process. Thus, it is natural to sample terms from paraphrases of the same question when generating keyword queries. We realize this idea by defining the term generation model $P(t_i|\theta_q)$ as a three component mixture:

$$P(t_i|\theta_q) = \alpha P(t_i|q) + \beta P(t_i|C_q) + (1-\alpha-\beta)P(t_i)\,, \qquad (6)$$

where C_q is a set of paraphrases of question q and $P(t_i|C_q)$ defines the likelihood of selecting term t_i from the paraphrases. All paraphrases in C_q are concatenated together into a single large document, then $P(t_i|C_q)$ may be calculated by one of three strategies we described in the previous section. The model in Eq. (6) has two parameters, $\alpha, \beta \in [0,1]$. As α tends to one, it assumes that the user definitely remembers the terms of the original question. As β tends to one, it assumes that user does not recall the terms from the original question but knows how to paraphrase it. As both α and β tend to zero, it means that user knows that the question exists but does not remember any terms from the original question nor from any of its paraphrases.

3.1.4 Phrase Detection. We sample not only terms but also phrases, in order to avoid breaking up continuous word sequences that constitute meaningful units. Specifically, we follow the method proposed by Mikolov et al. [16] for detecting phrases. Words that belong to the same phrase are grouped together into a new term. For example, the question *"how fast is a 2004 honda crf 230"* is converted to *"how fast is a 2004 honda_crf_230"* after phrase detection. This way, KQGM is able to directly sample *honda_crf_230*, instead of sampling three independent terms.

3.2 Filtering Mechanisms

To ensure that high-quality synthetic data is generated for training neural translation models, we propose two filtering mechanisms. One operates on the level of individual questions and selects the best keyword query, from a pool of candidate queries generated for a given question (§3.2.1). The other filter is applied over the entire set of synthetic query-question pairs and filters out low-quality instances (§3.2.2).

3.2.1 Keyword Query Filter. Given the probabilistic nature of query length selection (line 5 in Algorithm 1) and term selection (line 7 in Algorithm 1), the keyword query generation model may produce very different keyword queries for the same question. These keywords may vary a lot in terms of quality, from appropriate to inadequate. For example, given the question *"what happens inside a refracting telescope,"* the query generation model can give rise to a good keyword query, *"happens inside refracting telescope,"*

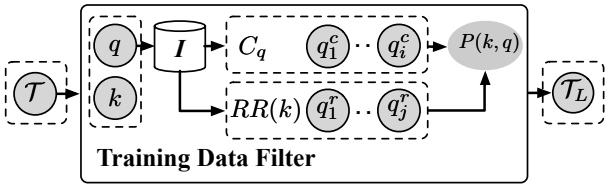

Figure 4: The architecture of our training data filter (TDF).

or to a rather bad one, *"inside colors type,"* using the very same parameters. The idea is to remedy this behavior by generating, for each question, a set of candidate keyword queries (i.e., running the model multiple times), and then selecting the single most suitable query. We propose to achieve this using a so called *keyword query filter* (KQF), shown in Fig. 3. The intuition behind this ranking-based filtering approach is that the better the generated keyword query is, the more effectively it can retrieve the original question from the question corpus. (It is worth pointing out that our algorithm will always generate a keyword query that is shorter than the corresponding question, i.e., it is never the same as the question.)

We start with generating a set of m candidate keywords $\mathcal{K} = \{k_1, \ldots, k_m\}$ for a given question q using KQGM. Then, we issue each candidate keyword query k_i against an index containing all questions in our corpus, and retrieve the top-N highest scoring questions, $R(k_i) = \langle q_1^i, q_2^i, \ldots, q_N^i \rangle$. Specifically, we employ the Sequential Dependence Model (SDM) retrieval method [15]. Finally, we select the best candidate keyword k for the input question q according to its reciprocal rank:

$$k = \arg\max_{i \in [1 \ldots m]} \frac{1}{rank(q, R(k_i))}\,, \qquad (7)$$

where $rank(q, R(k_i))$ is the rank of q in the ranked list $R(k_i)$.

3.2.2 Training Data Filter. Even after applying the keyword query filter, there may still exist low-quality training instances in $\mathcal{T}$, which would misdirect the learning process. Therefore, we propose a *training data filter* (TDF) to filter out low quality instances. TDF, shown in Fig. 4, takes a set of synthetic query-question pairs $\mathcal{T}$ as input, and returns a subset $\mathcal{T}_L \subseteq \mathcal{T}$ that contains the top-L pairs with the highest *quality score*. We use retrieval precision as a quality indicator, which expresses to what extent k is a proper keyword for question q:

$$P(k, q) = \frac{|C_q \cap RR(k)|}{|RR(k)|}\,, \qquad (8)$$

where $RR(k)$ denotes the set of relevant questions retrieved by the keyword query k using the SDM retrieval method [15], and C_q denotes the set of paraphrase questions for q. In short, TDF ranks all generated query-question pairs according to $P(k, q)$, then selects the top-L highest scoring ones to form the filtered subset $\mathcal{T}_L$.

4 DATA

Our approach needs a small set of hand-labeled keyword-question pairs and a large set of questions. We obtain these two datasets from WikiAnswers.[2] WikiAnswers includes millions of questions asked by humans. Users have also identified groups of questions

[2] http://knowitall.cs.washington.edu/oqa/data/wikianswers/

that are paraphrases of each other. These groups are considered paraphrase clusters [8].

Preprocessing. Since we only care about natural language questions in this work, we employ the heuristics proposed by Dror et al. [7] to filter out non-natural language questions. Specifically, we keep only questions that start with "WH words" or auxiliary verbs. Additionally, we restrict ourselves to questions consisting of 5-12 terms (most frequent query length), based on question length distribution statistics of WikiAnswers; we refer to the online appendix for further details.[3] We end up with 3,168,878 paraphrase clusters, with 26.05 questions per cluster on average. In the remainder of the paper, when we write WikiAnswers, we refer to this preprocessed subset of the collection.

Small Set of Keyword-Question Pairs ($\mathcal{T}_0$). In order to get the small set of hand-labeled keyword-question pairs, we randomly pick 200 clusters from the 3,168,878 paraphrase clusters. From each of those paraphrase clusters, we sample five questions randomly. We employ five human annotators, who each receive only one question from each of the 200 paraphrase clusters. The annotators then manually create keyword queries from their questions. We then have 200 paraphrase clusters, each with five questions' paraphrases and corresponding keyword queries (where each paraphrase is labeled by a different annotator), a total of 1000 hand-labeled pairs.

Large Set of Questions ($\mathcal{Q}$). To get the large set of questions, we randomly sample a single question from each of the remaining paraphrase clusters. This amounts to 3,168,678 questions. The hand-labeled questions do not appear in this set.

5 EXPERIMENTAL SETUP

This section details various settings of three main components used in our approach, i.e., KQGM, filtering mechanisms, and NMT.

Keyword Query Generation Model. The following settings are used in our experiments:

- *Query length*: The prior probability of query length $P(s)$ is calculated based on the small set of (hand-labeled) keyword-question pairs. According to statistics on user keyword queries from the Yahoo! L16 Webscope Dataset, most keyword queries contain between 3 and 7 terms (see online appendix). Thus, we only sample queries with length $s \in [3, 7]$.
- *Collection Language Model*: The collection language model probability of $P(t_i)$ is computed based on the WikiAnswers dataset. For the paraphrase-based model, we need to know the paraphrases C_q for a given question q. In our dataset, this is readily available. We note that there also exist methods to detect paraphrases automatically [5].
- *Parameter Tuning*: For the baseline model (§3.1.1), there is only one free parameter $\lambda \in [0.1, 0.9]$. The paraphrase-based model (§3.1.3) involves two parameters, $\alpha \in [0.1, 0.9]$ and $\beta \in [0.1, 1-\alpha]$. We set the parameter values by performing an extensive (grid) search in steps of 0.1.

Filtering Mechanisms. For filters, we use the following settings:

- *Keyword query filter* (§3.2.1): We generate $m = 20$ candidate keywords for each question q in the large set of questions using KQGM. The best of these is selected by KQF to be paired with q.
- *Training data filter* (§3.2.2): For a keyword-question pair (k, q) we retrieve the top $N = 100$ questions using k and obtain the paraphrases C_q from paraphrase cluster of q.

Neural Networks. We implement the following three networks:

- *EDNet*: Basic encoder-decoder NMT network [19].
- *AttNet*: EDNet with attention mechanism [3].
- *CopyNet*: AttNet plus copying mechanisms [10].

For all three networks, we choose the top 44K most frequent words in WikiAnswers as our vocabulary. We set the embedding dimension to 100, and initialize the word embeddings randomly with a uniform distribution in [-0.1,0.1]. We set the number of layers of both encoder and decoder RNNs to 1. Further, we use a bidirectional GRU [3] unit with size 200 for encoder RNNs, and a GRU unit with size 400 for decoder RNNs. All networks are optimized using Adam [12] with an initial learning rate of 10^{-4}, gradient clipping of 0.1, and dropout rate of 0.5.

5.1 Preliminary Study

Our synthetic data generation heavily depends on the generative model for creating keyword queries. Thus, we perform a preliminary study, using the small set of keyword-question pairs, $\mathcal{T}_0$, to analyze the performance of various KQGM configurations. Informed by this analysis, we can decide which of the three term selection strategies to use for KQGM in our main experiments.

We use automatic metrics from text summarization, specifically, the widely used ROUGE-L metric [14]. ROUGE-L not only awards credit to in-sequence unigram matches, but also captures word order in a natural way. Thus, it can effectively measure the degree of match between the synthetic and ground truth keyword queries. Recall that in our dataset, we have a set of paraphrases for each question. We wish to consider those paraphrases as well in our evaluation. Formally, let k denote the generated keyword query corresponding to question q; C_q denotes the paraphrase cluster of q; $\mathcal{K}_q$ is the set of ground truth keywords corresponding to C_q. For scoring k, we consider the set of ground truth keywords $\mathcal{K}_q$ in two different ways: (i) by computing the average ROUGE-L between k and each ground truth keyword $k' \in \mathcal{K}_q$ (Eq. (9)), and (ii) by considering only the best (highest scoring) ground truth keyword query (Eq. (10)).

$$AvgRougeL = \frac{\sum_{k' \in \mathcal{K}_q} RougeL(k, k')}{|\mathcal{K}_q|} \tag{9}$$

$$MaxRougeL = \max_{k' \in \mathcal{K}_q} RougeL(k, k') \tag{10}$$

We employ five-fold cross-validation for evaluation. To eliminate the effects of randomness that is involved in the process, we repeat 100 times, and report the means and standard deviations.

Table 1 shows the evaluation results for all KQGM configurations. Comparing the three term selection strategies (§3.1.1), we find that the *Combination* strategy always attains the best performance. With the same term selection strategy and KGQM, phrase detection brings noticeable improvements in both AvgRougeL and

[3] http://bit.ly/2yNNzR0

Table 1: Evaluation of various KQGM configurations.

Configuration	AvgRougeL	MaxRougeL
Baseline model		
Popular	0.1956 (0.0934)	0.3197 (0.1266)
Discrimination	0.1877 (0.1049)	0.2999 (0.1421)
Combination	0.2240 (0.0953)	0.3522 (0.1331)
Baseline model + phrase detection		
Popular	0.2069 (0.1008)	0.3354 (0.1342)
Discrimination	0.2062 (0.1106)	0.3243 (0.1465)
Combination	0.2373 (0.1019)	0.3708 (0.1399)
Paraphrase-based model		
Popular	0.2125 (0.0930)	0.3390 (0.1250)
Discrimination	0.2266 (0.1017)	0.3458 (0.1367)
Combination	0.2435 (0.0956)	0.3734 (0.1330)
Paraphrase-based model + phrase detection		
Popular	0.2182 (0.1001)	0.3476 (0.1322)
Discrimination	0.2355 (0.1020)	0.3513 (0.1361)
Combination	**0.2521** (0.1009)	**0.3843** (0.1374)

MaxRougeL (+5.28% and +4.22%, respectively). Comparing the paraphrase based model with the baseline model, the former brings +10.66% improvements on average for AvgRougeL and +7.16% on average for MaxRougeL. The paraphrase-based model with phrase detection achieves the best overall performance, with 0.2521 AvgRougeL and 0.3843 MaxRougeL, which is superior to the best baseline configuration. Besides, based on manual inspection of synthetic keyword-question pairs, we find that the most prominent flaws in our synthetic data are extraneous terms in the KQGM-made keywords. For example, given the question *"what is usage of erw pipe,"* our KQGM generates a keyword query *"erw pipe usage made meant,"* where "made meant" are unnecessary terms.

5.2 Implemented Systems

5.2.1 Baseline systems. We implement the SDM retrieval model [15] and the state-of-the-art template-based method (TBM) [7] as baselines. The template-based K2Q method requires millions of hand-labeled keyword-question pairs from a query log, which we do not have access to. Thus, we use our simulated keyword-question pairs instead of hand-labeled keyword-question pairs and compute term similarity using word2vec vectors, instead of TF-IDF weighted context vectors. For the baseline systems, we retrieve the best matching question for each keyword query.

5.2.2 Neural systems. We train a neural network model with synthetic data, then feed the keyword query into the trained neural network model, to generate the most probable question. Specifically, we use the best KQGM configuration (paraphrase-based model with combination selection strategy and phrase detection), along with the keyword query filter to generate synthetic data (a total of 3,168,678 keyword-question pairs). Then, we use the training data filter to rank all keyword-question pairs.

Table 2: Automatic evaluation results of baseline systems and three neural networks.

Method	ROUGE-L	ROUGE-1	ROUGE-2	BLEU
SDM	0.3650	0.4123	0.1940	0.2780
TBM	0.4357	0.5134	0.2056	0.2858
EDNet	0.4338	0.5236	0.2464	0.3045
AttNet	0.4945	0.5748	0.2877	0.3672
CopyNet	**0.5115**	**0.6074**	**0.3026**	**0.3718**

6 EXPERIMENTAL RESULTS

This section reports our evaluation results for the K2Q task. First, in Sect. 6.1, we measure the quality of the generated questions using machine translation metrics. Then, in Sect. 6.2, we employ human judges to assess a sample of questions along two dimensions: relevance and grammar.

6.1 Automatic Evaluation

We use $\mathcal{T}_0$ for the automatic evaluation of our K2Q methods, which comprises 1000 hand-labeled keyword-question pairs. Note that these keyword-question pairs have not been used for the training of neural K2Q models. Therefore, it is appropriate to use $\mathcal{T}_0$ as a test dataset. We report on widely-used machine translation metrics: BLEU [17] and different variants of ROUGE [14].

Table 2 presents the evaluation result for the baseline systems and for the three neural networks. Clearly, all NMT approaches perform better than the SDM baseline. As expected, the template-based method performs better than SDM, but it is still far behind CopyNet, which is the best neural method. Compared with the basic encoder-decoder NMT network, we find that the attention mechanism brings in noticeable improvements in ROUGE-L (+13.99%), ROUGE-1 (+9.78%), ROUGE-2 (+16.76%) and in BLEU (+20.59%) scores. Because of the extraneous terms issue (cf. §5.1) in our synthetic data, the attention mechanism plays a very important role in skipping those terms (by assigning small weights to extraneous terms in the decoding process). Additionally, the copying mechanism brings further minor improvements in ROUGE-L (+3.44%), ROUGE-1 (+5.67%), ROUGE-2 (+5.18%) and BLEU (+1.25%).

We seek to gain a better understanding of how the different elements of our synthetic data generation approach contribute to end-to-end performance on the K2Q task. For that reason, we train the best performing neural model (CopyNet) using different configurations for generating synthetic training data. We add components one by one, to see how they affect performance. Additionally, we vary the amount of training data used L between 0.5M and 3M pairs. The results are shown in Fig 5.

- *Baseline*: Baseline KQGM with the *Combination* term selection strategy (§3.1.1).
- *Par*: Paraphrase-base KQGM with the *Combination* term selection strategy (§3.1.3).
- *Par+Ph*: Phrase detection added on top (§3.1.4).
- *Par+Ph+KQF*: Keyword query filter added on top (§3.2.1).
- *Par+Ph+KQF+TDF*: Training data filter employed on top (§3.2.2).

The first three methods do not involve the keyword query filter. In those cases, we generate 20 candidate keyword queries for a

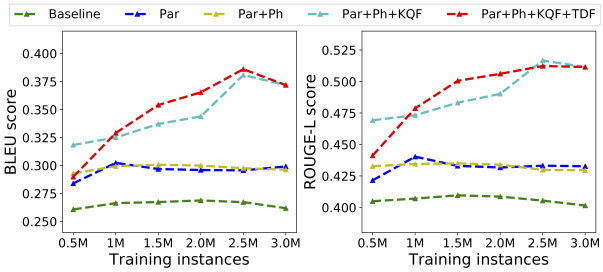

Figure 5: The influence of different components of our synthetic data generation approach on the end-to-end K2Q task. The x-axis represents the amount of training data (L); the y-axis indicates the BLEU/ROUGE-L score.

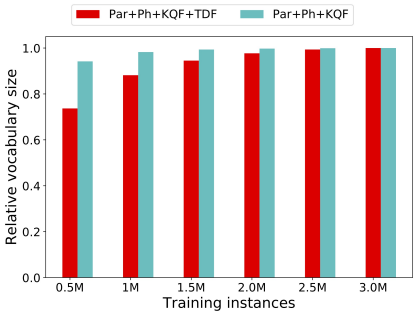

Figure 6: Fraction of the total vocabulary (y-axis) captured within the subset of training instances selected by TDF (x-axis). I.e., unique words present in $\mathcal{T}_L$, relative to $\mathcal{T}$.

given question and randomly select one of those. Only the last method uses TDF, which is a mechanism to select the top-L highest quality training instances (keyword-question pairs) into $\mathcal{T}_L$. For the other methods, we randomly select L instances from the entire synthetic training data set to form $\mathcal{T}_L$. We run methods that involve randomization three times and report the means.

From Fig. 5, we make the following observations. First, we find the results similar to that of the KQGM evaluation in Table 1. Among the three KQGMs, the *Par+Ph* model performs best. The paraphrase-based KQGM brings noticeable improvements compared to baseline-based KQGM in both ROUGE-L (+6.37% on average) and BLEU (+11.4% on average), while adding phrase detection on top of that only brings minor improvements in ROUGE-L (+0.13% on average) and BLEU (+0.71% on average).

Second, comparing the results of *Par+Ph* and *Par+Ph+KQF*, we find that the keyword query filter brings noticeable improvements in both ROUGE-L (+13.4% on average) and BLEU (+16.3% on average). Notice that by adding the keyword query filter, the performance of neural models improves with the size of the training data. Thus, the keyword query filter is an essential element in our synthetic data generation approach.

Third, we find that *Par+Ph+KQF+TDF* almost always performs better than *Par+Ph+KQF*, demonstrating that our training data filter is able to estimate the quality of the generated keyword-question pairs, and feed high-quality training instances into the neural networks. One noticeable exception (for both BLEU and ROUGE-L) is the leftmost data point ($L = 0.5M$), where the performance of *Par+Ph+KQF+TDF* is much below that of *Par+Ph+KQF*. A further analysis reveals that this is caused by an "insufficient vocabulary" issue. This is illustrated on Fig. 6, where we plot the fraction of the total vocabulary (i.e., unique words in $\mathcal{T}$) present in the training subset $\mathcal{T}_L$. We can observe that with only $0.5M$ training instances, the *Par+Ph+KQF+TDF* model has built up only 74% of the vocabulary, as opposed to 94% by the *Par+Ph+KQF* model. Our training data filter, based on a retrieval method, performs well with frequent terms, but fails on rare terms. It appears that the TDF quality score estimator overvalues common terms and undervalues rare terms, when selecting the subset of instances $\mathcal{T}_L$ for training.

Finally, as expected from TDF, it greatly benefits performance to use the high-quality training instances first; see the *Par+Ph+KQF+TDF* model for the 0.5M-1.5M range. In contrast, the last half million training instances yield little to no improvements. These results suggest that creating more high-quality keyword-question pairs might bring predictable improvements for neural K2Q models.

6.2 Manual Evaluation

We also perform a manual evaluation using a sample of 87 real user keyword queries with low query clarity[4] from the Yahoo! Webscope L16 Dataset. All these queries originate from the query log of Yahoo Answers. For each keyword query, we generate 5 questions, each with a different method. That is, the SDM and TBM baselines, and the three neural networks.

Three human raters were asked to provide score and assess each question along two dimensions: (i) *Relevance*, which indicates whether the question is relevant to the keyword content-wise (ignoring grammar mistakes), and (ii) *Grammar*, which reflects the grammatical correctness. Details of the rating scheme are included in our online appendix. Raters were further asked to choose the best generated question from among the five alternatives. The number of wins were then aggregated for each of the five methods. If multiple methods generated the same question, then the point is added to all. Table 3 shows the results of human judges; the reported scores are means. As expected, the SDM method scores highest on grammar, since it retrieves existing questions from the corpus. However, it achieves a very low score on relevance, since it can only retrieve questions that have been asked before (i.e., exist in the corpus). As in the automatic evaluation results, the attention mechanism brings in substantial improvements over the simple Encoder-Decoder model (both in terms of relevance and grammar). As anticipated, the copying mechanism leads to large improvements in terms of relevance (+40.3%); at the same time, the grammar score of CopyNet is only marginally lower than that of AttNet. Table 4 provides some examples of generated questions. Clearly, SDM returns grammatically correct, but often irrelevant questions. CopyNet has the ability to capture the meaning of the keyword query, and generates somewhat monotonous but very relevant questions. The other two neural networks seem to capture the query intent only partially, and drift off in directions that

[4]Query clarity ranges from 1.0 to 3.0, where 1.0 indicating "clear" and 3.0 indicating "vague." We only sample queries with clarity smaller than 1.5.

Table 3: Manual K2Q evaluation results. The inter-rater agreement is measured using Cohen's kappa score [6]. Highest scores are in boldface.

Methods	Relevance	Grammar	Wins
SDM	0.352	**1.643**	7.333
TBM	1.065	0.590	14.333
EDNet	0.569	0.682	6.666
AttNet	1.114	1.046	31.666
CopyNet	**1.563**	0.998	**36.000**
Cohen's kappa score	0.499	0.498	0.637

Table 4: Examples of generated questions from our K2Q system. The methods used to generate question are [S] SDM, [T] TBM, [E] EDNet, [A] AttNet, and [C] CopyNet.

Keyword 1 *cute yaoi animes*
[S] Do girls watch yaoi anime?
[T] Is it cute when yaoi are animes?
[E] Are there a good animes are cute?
[A] What are cute boobs?
[C] What are cute yaoi animes?
Keyword 2 *average price movie ticket 1987*
[S] What is the average ticket price for a super bowl ticket?
[T] What is the average price of movie ticket 1987?
[E] What is the average price for a 1987 ticket in 1987?
[A] What is the average price for a movie ticket?
[C] What is the average price of the movie ticket in 1987?
Keyword 3 *popular jbs england*
[S] How big are jbs feet?
[T] Who are popular sovereignty and jbs england related?
[E] What is the most popular in england?
[A] Who is popular in england?
[C] How popular is jbs in england?

are somewhat related to the topic of the query, yet irrelevant, e.g. "*What are cute boobs?*" and "*What is most popular in england?*"

7 CONCLUSIONS

In this work, we have studied the problem of translating keyword queries to natural language questions using neural approaches. To the best of our knowledge, this is the first application of neural machine translation methods to the keyword-to-question (K2Q) task. Perhaps the most innovative aspect of this work is the combination of keyword query generation models combined with various filtering mechanisms to create massive amounts of synthetic data for training neural models. Our empirical evaluation has demonstrated the effectiveness of our synthetic data generation approach for the K2Q task.

In this paper, we have generated only a single question for each keyword query, and evaluated it with respect to relevance and grammatical correctness. The same neural models, however, may also be used to generate a diverse list of questions for a given keyword query, with the help of techniques like beam search [9]. For example, given the keyword query "*Bible verse about education*," our neural models generated a range of diverse and meaningful questions, including:

- "*What is the fugitive slave verse about education?*"
- "*What is the christ verse about education?*"
- "*What is the sacred verse about education?*"
- "*What does Bible verse say about education?*"

In the future, we are interested in generating a diverse set of questions (i.e., the bottom part in Fig. 1) and comparing these with existing template-based methods with respect to diversity.

In summary, our methods have shown great potential and promise for creating synthetic training data that can be used to train robust neural models; future applications of this idea extend beyond the keyword-to-question task.

REFERENCES

[1] Leif Azzopardi and Maarten de Rijke. 2006. Automatic Construction of Known-item Finding Test Beds. In *Proc. of SIGIR '06*. 603–604.

[2] Leif Azzopardi, Maarten de Rijke, and Krisztian Balog. 2007. Building Simulated Queries for Known-item Topics: An Analysis Using Six European Languages. In *Proc. of SIGIR '07*. 455–462.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. *Neural Machine Translation by Jointly Learning to Align and Translate.* http://arxiv.org/abs/1409.0473.

[4] Richard Berendsen, Manos Tsagkias, Wouter Weerkamp, and Maarten de Rijke. 2013. Pseudo Test Collections for Training and Tuning Microblog Rankers. In *Proc. of SIGIR '13*. 53–62.

[5] Dasha Bogdanova, Cícero Nogueira dos Santos, Luciano Barbosa, and Bianca Zadrozny. 2015. Detecting Semantically Equivalent Questions in Online User Forums. In *Proc. of CoNLL '15*. 123–131.

[6] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[7] Gideon Dror, Yoelle Maarek, Avihai Mejer, and Idan Szpektor. 2013. From Query to Question in One Click: Suggesting Synthetic Questions to Searchers. In *Proc. of WWW '13*. 391–402.

[8] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2014. Open Question Answering over Curated and Extracted Knowledge Bases. In *Proc. of KDD '14*. 1156–1165.

[9] Markus Freitag and Yaser Al-Onaizan. 2017. *Beam Search Strategies for Neural Machine Translation.* http://arxiv.org/abs/1702.01806.

[10] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. *Incorporating Copying Mechanism in Sequence-to-Sequence Learning.* http://arxiv.org/abs/1603.06393.

[11] Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla. 2015. *SceneNet: Understanding Real World Indoor Scenes With Synthetic Data.* http://arxiv.org/abs/1511.07041.

[12] Diederik P. Kingma and Jimmy Ba. 2014. *Adam: A Method for Stochastic Optimization.* http://arxiv.org/abs/1412.6980.

[13] Alexander Kotov and ChengXiang Zhai. 2010. Towards Natural Question Guided Search. In *Proc. of WWW '10*. 541–550.

[14] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Proc. of the ACL '04 Workshop*. 74–81.

[15] Donald Metzler and W. Bruce Croft. 2005. A Markov Random Field Model for Term Dependencies. In *Proc. of SIGIR '05*. 472–479.

[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proc. of NIPS '13*. 3111–3119.

[17] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proc. of ACL '02*. 311–318.

[18] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. 2016. The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. In *Proc. of CVPR '16*.

[19] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. *Sequence to Sequence Learning with Neural Networks.* http://arxiv.org/abs/1409.3215.

[20] Shiqi Zhao, Haifeng Wang, Chao Li, Ting Liu, and Yi Guan. 2011. Automatically generating questions from queries for community based question answering. In *Proc. of IJCNLP '11*. 929–937.

[21] Zhicheng Zheng, Xiance Si, Edward Y Chang, and Xiaoyan Zhu. 2011. K2Q: Generating Natural Language Questions from Keywords with User Refinements.. In *Proc. of IJCNLP '11*. 947–955.