An End-to-End Generative Retrieval Method for Sponsored Search

Yijiang Lian*
Baidu
lianyijiang@baidu.com

Zhijie Chen
Baidu
bytechen.hit@gmail.com

Jing Jia
Baidu
jiajing01@baidu.com

Zhenjun You
Baidu
youzhenjun@baidu.com

Chao Tian
Baidu
tianchao@pku.edu.cn

Jinlong Hu
Baidu
hujinlong01@baidu.com

Kefeng Zhang
Baidu
zhangkefeng@baidu.com

Chunwei Yan
Baidu
yanchunwei@baidu.com

Muchenxuan Tong
Baidu
demon386@gmail.com

Wenying Han
Baidu
hanwenying@baidu.com

Hanju Guan
Baidu
guanhanju@baidu.com

Ying Li
Baidu
leeyingxj@gmail.com

Ying Cao
Baidu
yincao@microsoft.com

Yang Yu
Baidu
I@reyoung.me

Zhigang Li
Baidu
lizhigang01@baidu.com

Xiaochun Liu
Baidu
liuxiaochun@baidu.com

Yue Wang
Baidu
wangyue@baidu.com

ABSTRACT

In this paper, we propose an end-to-end generative retrieval method for sponsored search, which uses neural machine translation (NMT) to generate keywords directly from queries. To ensure that all the generated sentences are commercial keywords, a Trie-based pruning scheme is designed in the phase of decoding. Combined with self-normalization and dropping inferior candidates on the fly, 66% of the decoding time can be saved without degrading the relevance quality. Both organic and commercial click logs are used as parallel training data, which can and does encourage the model to generate more incremental keywords. To deploy this method as an online service, an online-offline mixing structure is devised to reduce latency and CPU consumption. This method has been successfully applied in Baidu's sponsored search, which has brought a significant revenue increase of more than 10%.

*Corresponding author

WWW '21, The Web Conference , April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CCS CONCEPTS

• **Information systems** → *Probabilistic retrieval models.*

KEYWORDS

sponsored search, keyword retrieval, generative model, beam search, prefix tree, information retrieval, neural machine translation

ACM Reference Format:
Yijiang Lian, Zhijie Chen, Jing Jia, Zhenjun You, Chao Tian, Jinlong Hu, Kefeng Zhang, Chunwei Yan, Muchenxuan Tong, Wenying Han, Hanju Guan, Ying Li, Ying Cao, Yang Yu, Zhigang Li, Xiaochun Liu, and Yue Wang. 2021. An End-to-End Generative Retrieval Method for Sponsored Search . In *WWW '21: The Web Conference , April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Sponsored search is an interplay of three entities. *Advertisers* provide business advertisements and bid for keywords to target their audience. *Search engines* provide platforms where *advertisers*' ads can be shown to *users* along with the organic results. *Users* submit queries to *search engines* and interact with ads. A modern sponsored search engine usually provides flexible match types such as *exact, phrase, broad* [1], with which the advertisers can specify how would their purchased keywords be matched to the online queries. In the *broad match* type scenario, an ad may be shown for a query if it is semantically relevant to the ad's keyword.

[1] https://support.google.com/google-ads/answer/7478529?hl=en

Generally, a sponsored search system consists of three modules: *keyword matching*, *ad retrieving*, and *ad selection*. In this paper, we focus on the *keyword matching* problem for the *broad match* type. One great challenge of this problem is the semantic gap between queries and keywords, since advertisers and users might describe the same thing in quite different ways. Besides, most ad keywords are short texts, which increases ambiguity and makes the gap even more serious. Most sponsored search systems use query rewriting technique [9, 13, 15, 17] to alleviate this problem, where the keyword matching process is divided into three separate stages: *query rewrite*, *boolean retrieval*, and *relevance filtering*. In a real industrial sponsored search system, these three stages are usually carefully tuned to balance effect and latency, which greatly limits the system's retrieval ability.

In this paper, we propose an end-to-end generative retrieval method (GRM) to improve the system's retrieval performance. In this method, a standard encoder-decoder neural machine translation structure is deployed, within which the query is encoded by a multi-layer Recurrent Neural Network (RNN) encoder into a list of hidden states, and then a multi-layer RNN decoder is used to decode the target keyword token by token based on these hidden states and the previously generated tokens. During inference, a beam search strategy is used to approximately generate the top k best translations.

The motivation of this method is to use translation model's generalization ability to produce more relevant keywords. A translation model is trained on a parallel corpus $\mathcal{D}$ of <query, keyword> pairs. Then we use this model to generate keywords for queries in $\mathcal{D}$. With a large beam size, new <query, keyword> pairs that do not exist in the training data would be generated. For example, in Figure 1, there are two training examples: <query1, keyword1> and <query2, keyword2>. Since query1 and query2 are quite similar, the trained model might generalize and transfer the keywords retrieved by query2 to query1, or even generate more relevant keywords. In this example, keyword2 is transferred into query1's triggered list, and keyword3 and keyword4 are newly generated keywords.

To carry out this idea in a real industrial environment is a challenging task. Firstly, decoding in sponsored search scenario is a constrained closed target domain problem, where only keywords committed by advertisers are permitted during the generation. A prefix tree is introduced into the decoding phase to fix this problem. Secondly, the decoding speed of a common beam search strategy is difficult to meet the requirements. Trie-based dynamic pruning combined with dropping inferior candidates on the fly is introduced to address this problem. Thirdly, it is difficult to deploy the NMT model as an online service due to the limited latency and computing resources. An online-offline mixing architecture is proposed to solve this problem.

This generative retrieval method has been successfully realized in Baidu's sponsored search. We hope this would shed light on the further design of sponsored retrieval system and NMT's application in industry.

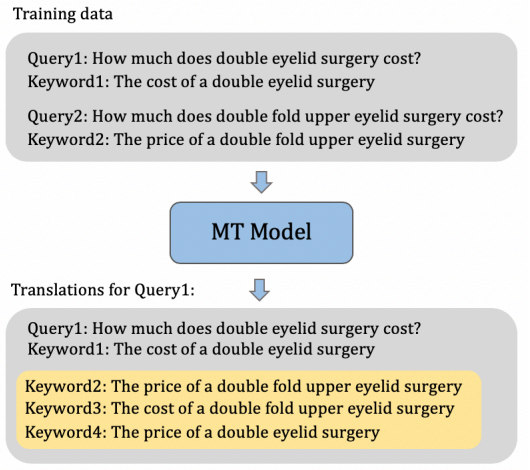

Figure 1: An example which can explain our motivation of our generative retrieval method.

2 RELATED WORK

Machine translation is a popular way to alleviate the semantic gap in information retrieval. With a parallel corpus, machine translation can learn the underlying word alignment between the target words and the source words. If we use monolingual parallel data, semantic synonymy can be detected. Basically, there are two kinds of applications of machine translation in information retrieval. The first one uses machine translation as a discriminative model to evaluate <query, doc> relevance. Given a query Q and a document D, the translation probability $P(D|Q)$ or $P(Q|D)$ was used as a feature to boost the calculation of query document relevance [3, 21]. Hillard et al. [6] applied this idea to calculate the commercial query ad relevance. The second one uses machine translation as a query rewriting method. Gao et al. [4], Jones et al. [10], Riezler and Liu [18] treated query rewrite as a statistical machine translation problem with monolingual training data. Recently, He et al. [5] proposed a sequence-to-sequence deep learning framework to study the query rewrite.

The most related work is the paper recently published by Lee et al. [14], which used conditional GAN to generate keywords from queries. There are several critical points that make our work different from theirs:

- The target domain in their translation setting is not closed. The generated sentence might not be a valid keyword.
- Unlike their approaches, organic and commercial click log are all used in our training data. This allows the NMT to generate more words not covered by the existing system.
- Random noise vectors are used in [14] to promote diversity, whereas our model is fully deterministic.
- Our work concentrates more on addressing the latency impact of deploying the generative model into the real commercial system. Nevertheless, they showed no experiment results in the industry environment.

Although NMT gives us a nice and simple end-to-end way to deploy a state-of-the-art machine translation system, its decoding

efficiency is still challenging. The standard beam search algorithm implemented by Bahdanau et al. [1] reduced the search space from exponential size to polynomial size, and was able to translate about ten words per second. Hu et al. [8] built a priority queue to further reduce the search space. And they also introduced a constrained softmax operation that uses a phrase-based translation system to generate the constrained word candidates. Since lots of unnecessary hypotheses are removed, the computational efficiency is greatly improved.

3 PROBLEM FORMULATION

In the following formulas, we use $\vec{q}$ to denote the vector of q, capital letters to represent sequences(e.g. Q, K), squiggle letters to represent set(e.g. $\mathcal{K}$) and lower case to represent individual tokens in a sequence (e.g. q_1, k_2), $k_{<i}$ to represent the token sequence $k_0, k_1, \ldots, k_{i-1}$, where k_0 is a special beginning of sentence symbol that is prepended to every target keyword.

Let (Q, K) be a <query, keyword> pair, where $Q = q_1, q_2, \ldots, q_M$ is the sequence of M tokens of the source query Q, and $K = k_1, k_2, \ldots, k_N$ is the sequence of N tokens in the target keyword K. From the probabilistic perspective, machine translation is equivalent to maximizing the log-likelihood of the conditional probability of sequence K given a source query Q, i.e. $\log P(K|Q)$, which can be decomposed into factors:

$$\log P(K|Q) = \sum_{i=1}^{N} \log P(k_i|k_{<i}; Q) \tag{1}$$

Our model follows the common sequence to sequence learning encoder-decoder framework [20] with attention [1]. Under this framework, an encoder reads the input query Q and encodes its meaning into a list of hidden vectors:

$$\vec{Q} = (\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_M) = \text{Encoder}(q_1, q_2, \ldots, q_M) \tag{2}$$

where $\vec{q}_i \in \mathbb{R}^n$ is a hidden state at time t. In our experiment, the encoder is mainly implemented by RNN:

$$\vec{q}_i = \text{RNN}(q_i, \vec{q}_{i-1}) \tag{3}$$

And the decoder is trained to predict the probability of next token k_i given the hidden states $\vec{Q} = (\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_M)$ and all the previously predicted words $k_1, \ldots, k_{i-1}$

$$P(k_i|k_{<i}; Q) \approx P(k_i|k_{<i}; \vec{Q}). \tag{4}$$

During inference, target tokens would be decoded one by one based on this distribution, until a special end of sentence symbol (<e>) is generated.

In order to focus on different parts of the source query during decoding, an attention mechanism [1] is introduced to connect the hidden states of decoder and encoder. Let k_{i-1} be the decoder output from the last decoding time step $i-1$, $\vec{c}_i$ be the attention context for the current time step i, which is calculated according to the following formulas:

$$\begin{aligned} \vec{c}_i &= \sum_{j=1}^{M} \alpha_{ij} \vec{q}_j, \\ \alpha_{ij} &= \frac{\exp(e_{ij})}{\sum_{p=1}^{M} \exp(e_{ip})}, \\ e_{ij} &= \text{Atten}(\vec{k}_{i-1}, \vec{q}_j) \end{aligned} \tag{5}$$

where Atten could be implemented as a dot product or feed forward network and $\vec{k}_i$ is the hidden state vector at time step i.

The RNN decoding phase is computed as follows:

$$\vec{k}_i = \text{RNN}(\vec{k}_{i-1}, k_{i-1}, \vec{c}_i) \tag{6}$$

$$s_i(w) = s(k_{i-1}, \vec{k}_i, \vec{c}_i, w) \tag{7}$$

$$p(k_i = w|k_{<i}; Q) = \frac{\exp(s_i(w))}{\sum_{w'} \exp(s_i(w'))} \tag{8}$$

where $s_i(w)$ is the unnormalized energy score of choosing k_i to be w.

4 METHOD

4.1 Trie-based Pruning

One challenge in applying neural machine translation to keyword retrieval task is that our target space is a restricted fixed set of submitted keywords, whereas in general translation, the target space is unconstrained. One possible method to mitigate this problem is to generate as many candidates as possible, then pick out the real keywords. However, this is quite inefficient, since NMT decoding is quite time-consuming.

In this paper, we devise a novel pruning technique in beam search called Trie-based pruning to fix this problem. Trie (also known as a prefix tree) is a popular data structure widely used in query auto-completion[7, 11]. In our scenario, a prefix tree $T_{\mathcal{K}}$ for the keyword repository $\mathcal{K}$ is built ahead of the decoding phase. First of all, each keyword K in $\mathcal{K}$ is tokenized into a token list, where a start symbol <s> is put at the front and an end symbol <e> is put at the end. Then we use these token lists to build a prefix tree keyed by tokens as is illustrated in Figure 2.

Suppose we are at the ith set of the decoding phase, $i-1$ tokens have been generated, the beam search size is B, and B hypotheses are kept in the set BeamSet = $\{\hat{K}^j_{i-1} = (k^j_1, k^j_2, \ldots, k^j_{i-1}), 1 \le j \le B\}$. With a prefix tree $T_{\mathcal{K}}$, we can get all the valid suffix tokens following the trie path $\hat{K}^j_{i-1}$. Only these valid suffix tokens would be considered in the probability inference stage of $p(k^j_i|k^j_1, \ldots, k^j_{i-1})$, all the others would be pruned away. Figure 3 shows the whole pruning process. Trie-based pruning technique guarantees that all the generated sentences are valid keywords, which greatly improves efficiency.

4.2 Self-normalization

One serious performance bottleneck at the NMT inference decoding stage is the computation of the denominator of softmax, i.e.

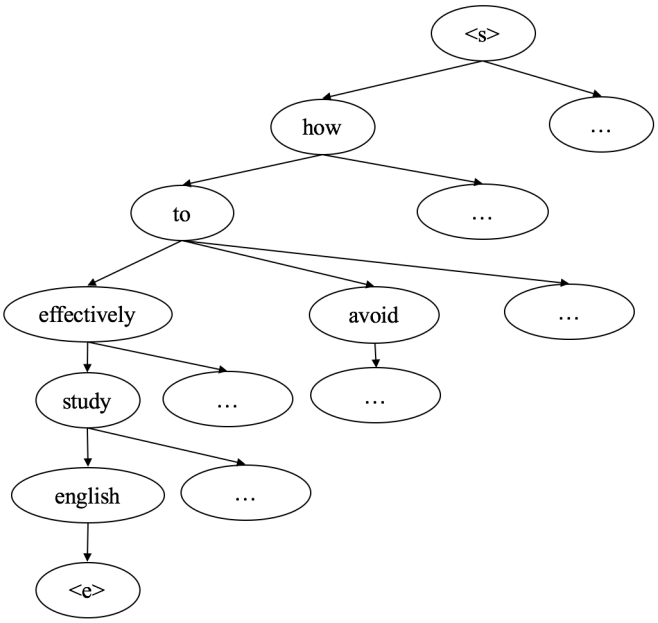

Figure 2: A schematic diagram for a prefix tree.

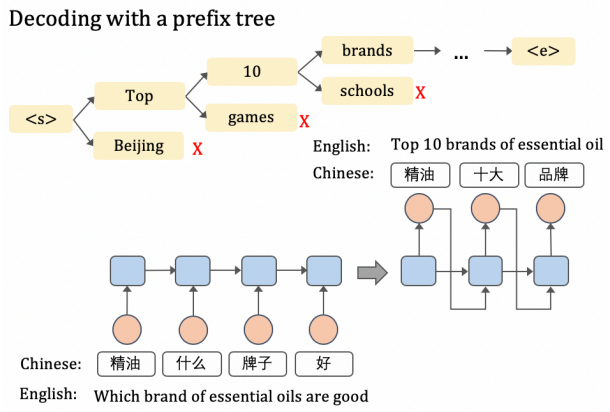

Figure 3: At the inference stage, we operate the beam search based on a prefix tree to decode the query into a closed keyword set.

$\sum_{w'} \exp(s_i(w'))$ in equation 8, as it involves summarizing over the entire output vocabulary space. Following Devlin et al. [2]'s work, we use the self-normalizing trick to speed up the decoding. To be specific, during training, an explicit regularization loss is added to the original likelihood loss in Equation 1 to encourage the softmax normalizer to be as close to 1 as possible.

$$\begin{aligned} L &= \sum_i \log(P(k_i|k_{<i}, Q)) - \beta(\log(\sum_{w'} \exp(s_i(w'))) - 0)^2 \\ &= \sum_i \log(P(k_i|k_{<i}, Q)) - \beta(\log(\sum_{w'} \exp(s_i(w'))))^2 \end{aligned} \tag{9}$$

When decoding with a self-normalized model, the costly step for calculating the denominator $\sum_{w'} \exp(s_i(w'))$ is avoided, we only have to compute the numerator $s_i(w)$. Furthermore, combined with a prefix tree, much more time can be saved, since only a small number of the valid suffix words have to be calculated,

4.3 Drop Inferior Hypotheses On The Fly

Another useful trick in our implementation is to remove the inferior hypotheses on the fly. Generally, a likelihood threshold is set to filter the final generated keywords at the end of decoding. This threshold can also be used in the internal process of decoding. As we decode a new token based on the current hypothesis, the likelihood of hypotheses would be multiplied by another probability factor, therefore the full likelihood decreases as decoding proceeds. Based on this consideration, if the current hypothesis's likelihood is lower than the given threshold, we would not expand it out later. This trick can generate more qualified keywords (with a likelihood above the threshold) in the final hypothesis set. Combined with the Trie-based pruning, the total decoding time would also be decreased.

4.4 Algorithm

Suppose $\mathcal{D}$ is the parallel training data set composed of <query, keyword> pairs, $\mathcal{K}$ is the keyword repository, which is a snapshot of all the keywords committed by advertisers, B is the beam size, $\mathcal{Q}$ is the query set. For each query in $\mathcal{Q}$, we want to get B relevant keywords in $\mathcal{K}$.

The whole process can be divided into three steps:

(1) Train a Self-normalized NMT model M with monolingual parallel data $\mathcal{D}$.
(2) Build a prefix tree T for the keyword repository $\mathcal{K}$.
(3) For each query in $\mathcal{Q}$, using M and T to decode out B relevant keywords in $\mathcal{K}$.
The decoding algorithm is shown in Algorithm 1.

4.5 An Online-Offline Mixing Architecture

It is well known that search queries are highly skewed and exhibit a power-law distribution [16, 19]. That is, at a fixed time, the most popular queries compose the head and torso of the curve, in other words, approximately 20% of queries occupy 80% of the query volume. Inspired by this idea, we design an online-offline mixing architecture (Figure 4) to deploy our GRM as an online service. Under this framework, the query volume is divided into two parts: frequent queries and infrequent queries. Frequent queries are collected from query logs in a period, and the size is in the tens of millions. All the other queries are considered infrequent. And we use different strategies to tackle these two flows.

For frequent queries, the retrieval process consists of two phases: preprocessing phase and online phase. In the preprocessing phase, our translation is done ahead of time, and the translated keywords for each query are saved in a lookup table. In the online phase, the translated keywords can be obtained immediately by looking up the table. And the preprocessing is repeated periodically to follow the changes in the query population and keyword supply over time. In this scenario, all computations are done in an offline mode, where complex models can be used to generate high quality keywords.

Algorithm 1: Beam Search with Trie-based pruning and dropping inferior hypotheses on the fly.

Input: Self-normalized NMT M, Keyword Prefix tree T, Beam Size B, score threshold s_{min}

/* Suppose the beam size is B, and the score threshold for dropping inferior hypotheses on the fly is s_{min}. */

Output: Keywords set Out

```
cur_buffer ← ∅   // to store all the current hypotheses that need to be extended
tmp_buffer ← ∅   // a temporal variable to be exchanged with cur_buffer
Out ← ∅                                  // to store the final results
put <s> into cur_buffer
while cur_buffer is not empty and size(Out) < B do
    for each hypothesis c in cur_buffer do
        /* each hypothesis c in cur_buffer would be extended */
        get the valid suffix word set S_c for c with T
        /* Using prefix tree T to get the suffix word set S_c */
        for each suffix word w in S_c do
            extend partial hypothesis c with w to get new hypothesis c̃ = [c; w]
            using M to inference the hidden state vector defined in Equation 6
            calculate score s_c̃ for c̃ according to Equation 7
            if s_c̃ > s_min then
                /* c̃ is a good hypothesis */
                if w == <e> then
                    /* c̃ has been decoded completely */
                    put c̃ into Out
                end
                else
                    /* c̃ needs to be extended later */
                    put c̃ into tmp_buffer
                end
            end
        end
    end
    /* sort and get the top candidates in tmp_buffer */
    sort elements c̃ in the tmp_buffer according to their score s_c̃ and keep only the top B− size(Out).
    /* switch the two buffers */
    cur_buffer ← tmp_buffer
    tmp_buffer ← ∅
end
return Out
```

For infrequent ad-hoc queries, the translation has to be conducted fully from scratch. Due to limited latency, we implemented a simple model.

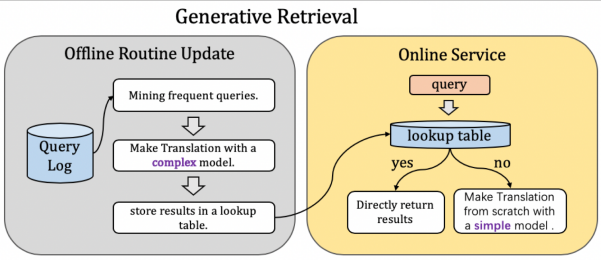

Figure 4: To deploy the GRM as an online service, an online-offline mixing architecture is designed.

5 OFFLINE EXPERIMENTS

5.1 Setup

Dataset. The training data is sampled from one month's click web log of the search engine, which comprises two parts: organic web log's <query, title> pairs and the sponsored web log's <query, keyword> pairs. And the size of the training data is about 500 million. 1000 queries and their corresponding clicked keywords have been sampled from the sponsored click log to form the testing dataset.

Vocabulary. The vocabulary contains 100,000 tokens which are the most frequent in the training dataset.

Prefix Tree. The prefix tree is built on the keyword repository. As shown in Table 1, the average number of suffixes decreases sharply as the depth of the trie increases. This confirms our motivation for utilizing a prefix tree to prune the beam search.

Table 1: Statistics of Suffix Number

Level	Mean Suffix Number
0	91100
1	73.72
2	3.62
3	1.77
4	1.35
5	1.17
6	1.09
7	1.05

5.2 Implementation Details

The offline experiments are run on a machine equipped with a 12-core Intel(R) Xeon(R) E5-2620 v3 clocked at 2.40GHz, a RAM of 256G, and 8 Tesla K40m GPUs.

The NMT model is implemented with a four-layer LSTM encoder and a four-layer LSTM decoder with an attention mechanism, whose encoder and decoder hidden size are both set to 512. The parameter β in equation 9 is carefully chosen to be 0.01 by grid search. And the loss function is minimized with an initial learning rate of 0.0005 by Adam[12] with a batch size of 128. With the trained NMT model, decoding is conducted on the testing dataset with a beam size of 30.

Three kinds of decoding methods are implemented for comparison, which are abbreviated as follows:

- *ST* is the standard beam search method.
- *TP* denotes the trie-based pruning method.
- *DropOTF + TP* denotes the strategy of dropping inferior hypotheses on the fly in the trie-based pruning method. The dropping threshold is set to -6.0.

5.3 Decoding Time Analysis

The first two columns in Table 2 show the result of decoding time comparison: *TP* reduces the total decoding time by around 45 percent, *DropOTF + TP* reduces it by about 66 percent. This improvement of decoding efficiency contributes a lot to the acceleration of the offline routine update presented in Figure 4.

For a detailed analysis, three key steps in beam search are selected.

(1) *Inference* step represents the computation of the hidden state vector defined in Equation 6, which corresponds to line 10 in Algorithm 1.
(2) *Score* step represents getting scores for candidate tokens. For *ST*, it corresponds to getting unnormalized scores in Equation 7 and then operating softmax in Equation 8. For *TP* and *DropOTF + TP*, the softmax operation is avoided by self-normalization and the *score* step means getting the valid suffix tokens' scores defined in Equation 7, corresponding to line 11 in Algorithm 1.
(3) *TopK* step represents sorting and getting top k candidates, which corresponds to line 22 in Algorithm 1.

Time consumed in these steps is respectively recorded.

As is shown in Table 2's last three columns, it's surprising that the reduction of time mainly comes from *TopK* instead of *Score*. This could be explained by the fact that: *TopK* contains a lot of CPU-intensive operations(like sorting), which results in a large amount of time spent on data communication between GPU and CPU. This makes it the most dominant time-consuming part of the whole process. *TP* and *DropOTF + TP* can effectively reduce the number of candidate hypotheses(which means the iterations of line 6 in Algorithm 1 would be reduced), thus greatly reducing the time consumed by *TopK*. On the other hand, the *Score* step does not take much time because the softmax operation runs on GPUs which is highly efficient in matrix multiply.

Table 2: Average time spent totally and average time spent on critical steps (in millisecond) for decoding a query by different beam search strategies.

Beam Search	Total	Inference	Score	TopK
ST	598.993	108.382	11.198	440.351
TP	330.395	107.506	7.482	95.965
DropOTF + TP	203.395	90.663	5.933	73.970

5.4 Relevance Evaluation

With the same trained model, the three decoding strategies mentioned above are used to generate keywords for queries in the testing Dataset. For each strategy, 400 query-keyword pairs are sampled from the generated results and sent to professional human judges for three grade labels: good, fair, and bad.

As is shown in Table 3, the *TP* method has reduced the bad case proportion from 18.0% to 9.7%, and *DropOTF + TP* further reduced it to 7.2%. This demonstrates that: under the condition of a great speedup, our method can still generate high-quality keywords. The BLEU results further confirm our conclusion.

Table 3: Query-keyword relevance evaluation for the decoding result generated by different beam search strategies. The H-xxx columns indicate the proportions of three grade levels judged by humans.

Beam Search	H-Good %	H-Fair %	H-Bad %	BLEU
ST	38.9	43.0	18.0	33.0
TP	49.3	41.0	9.7	33.6
DropOTF + TP	52.5	40.3	7.2	33.8

5.5 Delta Evaluation

Our main concern is whether the GRM model can generate a significant number of candidates that have not been retrieved by the current system. Therefore, we evaluate the Δ between keywords retrieved by models trained on different training datasets.

To make more diversified results, <query, title> pairs in organic click data have also been introduced as the training data. Although titles are quite different from keywords, which might result in a low validity of the generated keywords, *TP* can fix this problem. The experiment in Figure 5 shows the necessity of using *Trie-based Pruning*. For a model trained with organic click data, only a small number of the *ST* generated sentences are valid keywords. As the beam size increases, the number of valid keywords increases quite slowly. Especially, when the beam size is set as large as 300, only 8% of the results are valid keywords.

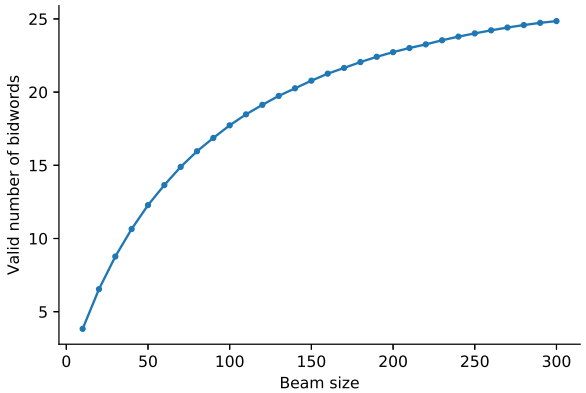

Figure 5: For a model trained with organic <query, title> click data, only a small number of the results produced by *ST* are actual keywords .

There are three sets for comparison:

- *Current* represents all the <query, keyword> pairs in the testing dataset.
- *Sponsor* represents the translation result produced by a model trained with sponsored click data.
- *Organic* represents the translation result produced by a model trained with organic click data.

Both *Sponsor* and *Organic* use *DropOTF + TP* as their decoding scheme.

Suppose the comparing set is $\mathcal{R}$, and the base set is $\mathcal{T}$, we define the evaluation metrics as follows:

$$\Delta = \frac{|\mathcal{R} - \mathcal{T}|}{|\{queries\ in\ \mathcal{T}\}|}$$

$$Ratio = \frac{|\mathcal{R} - \mathcal{T}|}{|\mathcal{T}|}$$

As is shown in table 4, the NMT model has a very good generalization ability that using sponsored data alone can bring about great Δ. With organic data, more and extra Δ can be got.

Table 4: Evaluation.

Comparing set	Base set	Δ	*Ratio*(%)
Sponsor	*Current*	26.42	1253
Organic	*Current*	28.3	1270
Organic	*Sponsor*	24.01	80

5.6 DropOTF Threshold Selection

The above experiments have shown that *DropOTF + TP* can reduce the decoding time greatly without compromising quality. However, the selection of the threshold for dropping inferior candidates is a trade-off among efficiency, quality, and quantity of results.

Table 5 shows that: as the threshold increases, the decoding time and the number of output keywords decrease, since more candidates are dropped in the beam search process. The BLEU value increases with the threshold for the same reason.

Table 5: The selection of the *DropOTF* threshold. The last column denotes the average number of generated keywords.

Threshold	Decoding Time(ms)	BLEU	Keywords Number
-6.5	221.369	33.6	28.5
-6.0	203.395	33.8	27.6
-5.5	184.045	34.3	26.2
-5.0	149.851	35.1	23.6
-4.5	119.557	36.1	19.7
-4.0	100.174	37.4	14.9

5.7 Case Study

Table 6 shows some typical results generated by GRM. From these cases, we can see that:

- The GRM model has great generalization ability. For query *Restaurant recommendation in Huimin Street*, only one <query, keyword> instance is found in the training data, which is <*Restaurant recommendation in Huimin Street*, *Huimin Street snacks*>. However, our model can generate two extra high-quality keywords: *Which restaurant in Huimin Street is delicious* and *Huimin Street Food Guide*. This further confirms our motivation mentioned in the introduction.
- In most cases, semantic meanings have been learned by the translation model. And the model is capable of generating lots of paraphrases. For example, keyword *Which place is suitable for travel in summer* perfectly matches query *Places worth visiting in summer*.
- The semantic relevance of different entities has been captured, which can be shown in this case: keyword *How do you like XiaoXiong thermal lunch box* is generated for query *How do you like Tiger thermal lunch box*.
- However, sometimes GRM might misunderstand the query and generate irrelevant results. For example, *(怎么炒黄金, How to invest in gold)* was generated for query *(黄金鲽鱼的做法, Cooking methods for Yellowfin Sole)*. In Chinese, these two sentences share the same term 黄金, and the character 炒 has diverse meanings such as investing and cooking. This polysemous phenomenon may make it difficult for the model to capture its semantic meaning. Another reason might be that there are not many similar queries in the training data.

6 ONLINE EXPERIMENTS

As mentioned in subsection 4.5, our GRM service is implemented with a mixing online-offline architecture. To be specific, for frequent queries (with a size of tens of millions), their GRM-triggered keywords are decoded with GPU by a large model mentioned in the offline experiment and these results are saved in a Key-Value table.

For infrequent queries, corresponding GRM keywords are decoded with CPU from scratch by a simple model, which is a single layer GRU encoder and a single layer GRU decoder both with a hidden size of 128. To guarantee the keywords' quality, online relevance judgment is conducted after translation. This mixed framework helps us to save more than 70% of CPU resources.

We use two metrics to evaluate the performance of our GRM service.

• CPM:CPM denotes revenue received by search engine for 1000 searches, which can be formalized as $\frac{\text{revenue}}{\#\{\text{searches}\}} \times 1000$.

• CTR:CTR denotes the average click ratio received by the search engine, which can be formalized as $\frac{\#\{\text{clicks}\}}{\#\{\text{searches}\}}$(one search means one submit of a query).

Table 7 shows the online A/B test result for the GRM service. The GRM system has contributed to a CPM growth of 13.8%, which is a dramatic improvement. CTR has increased by 15.4%, which demonstrates that the GRM does create a lot of new links for the underlying relevant query-keyword pairs, which brings about a good deal of user clicks.

7 CONCLUSIONS

In this paper, we have proposed a novel generative retrieval method for sponsored search. The motivation is that with a larger beam size, the seq2seq model might generate incremental keywords that can not be retrieved by the current system. To make the decoded

Table 6: Some typical keywords generated by GRM model.

Query	Generated Keywords	Label
回民街饭店推荐	回民街美食攻略	Good
Restaurant recommendation in Huimin Street	Huimin Street Food Guide	Good
回民街饭店推荐	回民街哪家好吃	Good
Restaurant recommendation in Huimin Street	Which restaurant in Huimin Street is delicious	Good
女生什么时候发育	女孩发育时间表	Good
When do girls start developing physically	Timing and stages of girls' physical development	Good
女生什么时候发育	女孩子多大开始长个子	Fair
When do girls start developing physically	When do girls start growing in height	Fair
虎牌保温饭盒怎么样	小熊保温饭盒怎么样	Fair
How do you like Tiger thermal lunch box	How do you like Xiaoxiong thermal lunch box	Fair
夏天值得一游的地方	夏天适合去哪里旅游	Good
Places worth visiting in summer	Which place is suitable for travel in summer	Good
黄金鲽鱼的做法	怎么炒黄金	Bad
Cooking methods for Yellowfin Sole	How to invest in gold	Bad

Table 7: Online A/B Test performance of the GRM system.

Indicator	Improvements
CPM	+13.8%
CTR	+15.4%

sentences are all valid keywords, a Trie-based pruning mechanism has been introduced. Trie-based pruning coupled with dropping inferior candidates on the fly saves 66% of the decoding time without degrading the relevance quality. Further, taking advantage of the power-law distribution of queries, a mixed online-offline architecture has been constructed, which makes GRM a real industrial service. This method has been successfully applied in Baidu's commercial search engine, which has generated a significant number of incremental keywords, resulting in a substantial revenue increase by more than 10%.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (ICLR)* (2015).

[2] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard M. Schwartz, and John Makhoul. 2014. Fast and Robust Neural Network Joint Models for Statistical Machine Translation. (2014).

[3] Jianfeng Gao, Xiaodong He, and Jian-Yun Nie. 2010. Clickthrough-Based Translation Models for Web Search: from Word Models to Phrase Models.

[4] 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* (Jeju Island, Korea) *(EMNLP-CoNLL '12)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 666–676.

[5] Yunlong He, Jiliang Tang, Hua Ouyang, Changsung Kang, Dawei Yin, and Yi Chang. 2016. Learning to Rewrite Queries. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (Indianapolis, Indiana, USA) *(CIKM '16)*. ACM, New York, NY, USA, 1443–1452.

[6] Dustin Hillard, Stefan Schroedl, Eren Manavoglu, Hema Raghavan, and Chirs Leggetter. 2010. Improving Ad Relevance in Sponsored Search. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining* (New York, New York, USA) *(WSDM '10)*. ACM, New York, NY, USA, 361–370.

[7] Bo-June Paul Hsu and Giuseppe Ottaviano. 2013. Space-efficient data structures for top-k completion. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 583–594.

[8] Xiaoguang Hu, Wei Li, Xiang Lan, Hua Wu, and Haifeng Wang. 2015. Improved beam search with constrained softmax for NMT. *Proceedings of MT Summit XV* (2015), 297.

[9] 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.

[10] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. 2006. Generating Query Substitutions. In *Proceedings of the 15th International Conference on World Wide Web* (Edinburgh, Scotland) *(WWW '06)*. ACM, New York, NY, USA, 387–396.

[11] Dimitrios Kastrinakis and Yannis Tzitzikas. 2010. Advancing search query autocompletion services with more and better suggestions. In *International Conference on Web Engineering*. Springer, 35–49.

[12] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)* (2015).

[13] Arnd Christian König, Kenneth Church, and Martin Markov. 2009. A data structure for sponsored search. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 90–101.

[14] Mu-Chu Lee, Bin Gao, and Ruofei Zhang. 2018. Rare Query Expansion Through Generative Adversarial Networks in Search Advertising. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) *(KDD '18)*. ACM, New York, NY, USA, 500–508.

[15] Azarakhsh Malekian, Chi-Chao Chang, Ravi Kumar, and Grant Wang. 2008. Optimizing Query Rewrites for Keyword-based Advertising. In *Proceedings of the 9th ACM Conference on Electronic Commerce* (Chicago, Il, USA) *(EC '08)*. ACM, New York, NY, USA, 10–19.

[16] Casper Petersen, Jakob Grue Simonsen, and Christina Lioma. 2016. Power Law Distributions in Information Retrieval. *ACM Trans. Inf. Syst.* 34, 2, Article 8 (Feb. 2016), 37 pages.

[17] Filip Radlinski, Andrei Broder, Peter Ciccolo, Evgeniy Gabrilovich, Vanja Josifovski, and Lance Riedel. 2008. Optimizing relevance and revenue in ad search: a query substitution approach. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 403–410.

[18] Stefan Riezler and Yi Liu. 2010. Query Rewriting Using Monolingual Statistical Machine Translation. *Comput. Linguist.* 36, 3 (Sept. 2010), 569–582.

[19] Amanda Spink, Dietmar Wolfram, Major B. J. Jansen, and Tefko Saracevic. 2001. Searching the web: The public and their queries. *Journal of the American Society for Information Science and Technology* 52, 3 (2001), 226–234.

[20] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3104–3112.

[21] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, Jean-Marc Langlois, and Yi Chang. 2016. Ranking Relevance in Yahoo Search. (2016), 323–332.