Context-Aware Query Rewriting for Improving Users' Search Experience on E-commerce Websites

Simiao Zuo[◇], **Qingyu Yin**[*†], **Haoming Jiang**[†], **Shaohui Xi**[†], **Bing Yin**[†], **Chao Zhang**[◇] and **Tuo Zhao**[◇]
{simiaozuo, chaozhang, tourzhao}@gatech.edu
{qingyy, jhaoming, shaohux, alexbyin}@amazon.com
[◇]Georgia Institute of Technology [†]Amazon

Abstract

E-commerce queries are often short and ambiguous. Consequently, query understanding often uses query rewriting to disambiguate user-input queries. While using e-commerce search tools, users tend to enter multiple searches, which we call context, before purchasing. These history searches contain contextual insights about users' true shopping intents. Therefore, modeling such contextual information is critical to a better query rewriting model. However, existing query rewriting models ignore users' history behaviors and consider only the instant search query, which is often a short string offering limited information about the true shopping intent. We propose an end-to-end context-aware query rewriting model to bridge this gap, which takes the search context into account. Specifically, our model builds a session graph using the history search queries and their contained words. We then employ a graph attention mechanism that models cross-query relations and computes contextual information of the session. The model subsequently calculates session representations by combining the contextual information with the instant search query using an aggregation network. The session representations are then decoded to generate rewritten queries. Empirically, we demonstrate the superiority of our method to state-of-the-art approaches under various metrics.

1 Introduction

Query rewriting is a task where a user inputs a potentially problematic query (e.g., typos or insufficient information), and we rewrite it to a new one that better matches the user's real shopping intent. This task plays an important role in e-commerce query understanding, where without proper rewriting, search engines often return undesired items, rendering the search experience unsatisfactory.

One major issue that impedes query rewriting is the ambiguity of queries. For example, Figure 1 (left) demonstrates searching for "bumblebee costumes" without considering search context. From the query alone, it is implausible to tell if the user's intent is for costumes of actual bumblebee (i.e., the animal) or the character from the movie franchise. This type of ambiguity is common in e-commerce search, where queries are usually short (only 2-3 terms) and insufficiently informative (He et al., 2016b). Therefore, it is not possible to disambiguate queries using only the instant search. A common solution is to use statistical rules to differentiate the possible choices. Specifically, in our example, suppose a total of 100 users entered the "bumblebee costumes" query, and 70 of them eventually purchased the movie character costume. When a new user searches for the same query, the recommended products will consist of 70% movie character costumes and 30% animal costumes. This procedure is problematic because each user has a specific intent, i.e., either the movie character costume or the animal costume, but rarely both, which the aforementioned method fails to address.

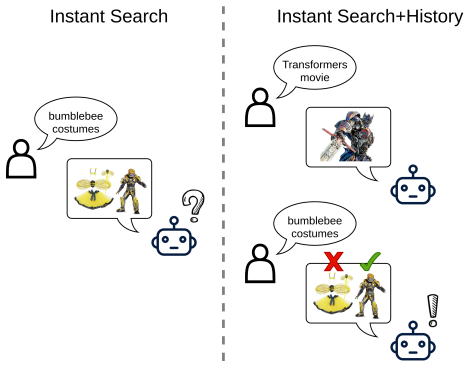


Figure 1: Searching for "bumblebee costumes" with (right) and without (left) history searches.

We propose to explore contextual information from users' history searches to resolve the query ambiguity issue. Taking the "bumblebee costumes" example again, in Figure 1 (right), suppose a rewriting model recognizes that the user searched for "Transformers movie" earlier, then it could infer

[*]Correspondence to Qingyu Yin (qingyy@amazon.com).

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 5: Industry Track, pages 616–628

that the user's purchase intent is the movie character costume, and hence can remove the input ambiguity. There have been existing works that utilize search logs for query rewriting. For example, Wang and Zhai (2007, 2008) use traditional TF-IDF-based similarity metrics to capture relational information among the user's history searches. These approaches are too restrictive to handle the increasingly complex corpus nowadays. As such, the rewritten queries significantly differ from the original one in intent. More recently, neural network-based query rewriting algorithms (He et al., 2016b; Xiao et al., 2019; Yang et al., 2019) are proposed. Most of such approaches employ a multi-stage training approach. Consequently, they involve complicated hand-crafted features or require excessive human annotations for the intermediate features (sometimes both).

To overcome the drawbacks of existing methods, we propose an end-to-end context-aware query rewriting algorithm. Our model's backbone is the Transformer (Vaswani et al., 2017). In our context-aware model, the Transformer encoder learns representations for individual history queries. The representations are further transformed to carry cross-query relational information using a graph attention mechanism (GAT, Velickovic et al. 2018). The GAT computes contextual information of a session based on a session graph, where its nodes contain the history queries and the tokens contained in the history queries. After obtaining the contextual information from the GAT, it is aggregated with the instant search using an aggregation network. The augmented information is subsequently fed into the Transformer decoder to generate rewritten queries.

Our proposed method improves upon existing works from three aspects. First, our model does not involve recursion, unlike conventional recurrent neural network-based approaches (He et al., 2016b; Yang et al., 2019; Xiao et al., 2019). This facilitates training deep models containing dozens of layers capable of capturing high-order information. Second, our end-to-end sequence-to-sequence learning formulation eliminates the necessity of excessive labeled data. Previous approaches (Yang et al., 2019; Xiao et al., 2019) require the judgment of "semantic similarity", and thus crave for human annotations, which are expensive to obtain. In contrast, our method uses search logs as supervision, which does not involve human effort, and are cheap to acquire. Third, our method can leverage powerful pre-trained language models, such as BART (Lewis et al., 2020). Such models contain rich semantic information and are successful in numerous natural language processing tasks (Devlin et al., 2019; Liu et al., 2019; Radford et al., 2019).

We demonstrate the effectiveness of our method on in-house data from an online shopping platform. Our context-aware query rewriting model outperforms various baselines by large margins. Notably, comparing with the best baseline method (Transformer-based model), our model achieves 11.6% improvement under the MRR (Mean Reciprocal Rank) metric and 20.1% improvement under the HIT@16 metric (a hit rate metric). We further verify the effectiveness of our approach by conducting online A/B tests.

2 Related Works

One line of work uses statistical methods. For example, Cui et al. (2002, 2003) extract probabilistic correlations between the search queries and the product descriptions. Other works extract features that are related to the user's current search (Huang et al., 2003; Huang and Efthimiadis, 2009), or from relational information among the user's history searches (Billerbeck et al., 2003; Baeza-Yates and Tiberi, 2007; Wang and Zhai, 2007; Cao et al., 2008; Wang and Zhai, 2008). There are also statistical machine translation-based models (Riezler et al., 2007; Riezler and Liu, 2010) that employ sequence-to-sequence approaches. The aforementioned statistical methods suffer from unreliable extracted features, such that the rewritten queries differ from the original one in intent.

Another line of work focuses on neural query rewriting models (He et al., 2016b; Xiao et al., 2019; Yang et al., 2019). These models adopt recurrent neural networks (RNNs, Hochreiter and Schmidhuber 1997; Sutskever et al. 2014) to learn a vectorized representation for the user's search query, after which KNN-based methods are used to find queries that yield similar representations. One major limitation is that the rewritten queries are limited to the previously presented ones. Also, these methods often involve complicated and ungrounded feature function designs, e.g., He et al. (2016b) and Xiao et al. (2019) hand-crafted 18 feature functions, or require excessive labeled data (Yang et al., 2019). Other works (Sordoni et al., 2015; Dehghani et al., 2017; Jiang and Wang, 2018) use RNNs for generative query suggestion,

but they inherit the weaknesses of RNNs and yield unsatisfactory performance in practice.

Note that Grbovic et al. (2015) construct context-aware query embeddings using word2vec (Mikolov et al., 2013). In their approach, an embedding is learned for each distinct query in the dataset. As such, the quality of the learned embeddings rely heavily on the number of occurrences of each query. This method is not applicable to our case because in our dataset, almost all the queries are distinct.

3 Problem Setup

The session data are collected from search logs. First, we collect all the searches from a specific user within a time window, and we call the searches a "session". After the user purchases a product, the session ends, i.e., we do not consider subsequent queries and behaviors after a purchase happens. This is because after a purchase, the user's intent often change. Note that different sessions may be collected from different users.

Each session contains multiple searches from the same user. We call the last query in the session the "target" query, the second to the last query the "source" (or the "instance) query, and the others the "history" queries. The intuition behind this is that because sessions always end with a purchase, the last search (i.e., the target) reflects the user's real intent. When the user enters the second to the last search (i.e., the source), if we can rewrite it to the target query, the user's intent will be fulfilled.

Below is an example of a search session. From the history queries, the user is interested in car related banners/posters. The source query contains a typo ("doger" is a baseball team) and we should rewrite it to the target query ("dodge posters").

History: {dodge banners; mopar poster}

Source (Instance): dodger posters

Target: dodge posters

We collect about 3 million (M) sessions, where each session consists of at least 3 history queries, a source query (i.e., the one we need to rewrite), and a target query (i.e., the ground-truth query that is associated with the purchase). We have roughly 18.7M queries, and on average, each session contains 4 history queries. Query rewriting is consequently formulated as a sequence-to-sequence learning problem. We highlight that per our formulation, we do not need human annotations, unlike existing approaches.

4 Method

Figure 2 illustrates our context-aware query rewriting model. The model contains four parts: a conventional Transformer (Vaswani et al., 2017) encoder, a graph attention mechanism (Velickovic et al., 2018) that captures the user's purchase intent, an aggregation network that encodes the history searches, and a conventional Transformer decoder that generates the rewritten query candidates.

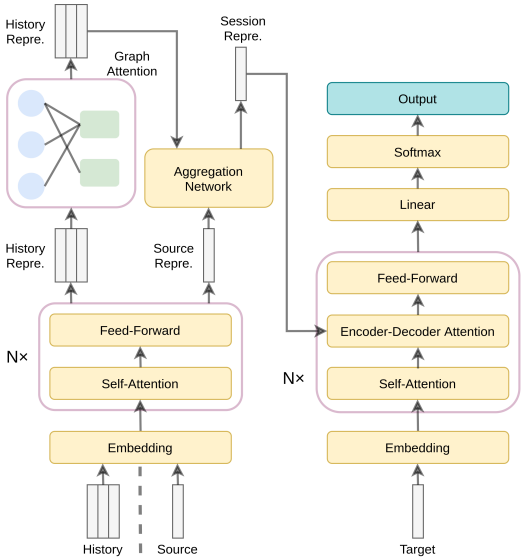


Figure 2: Overview of model.

4.1 Transformer Encoder

For a given source query, we first pad it with a <boq> (begin-of-query) token. Then, we pass the padded query through a Transformer encoder, after which we have its hidden representation $H_s \in \mathbb{R}^{L_s \times d}$. Here L_s is the length of the padded query, and d is the hidden dimension. We also pass all the history queries corresponding to this source query through the encoder, and we have the history query representation $U_h \in \mathbb{R}^{N_h \times L_h \times d}$, where N_h is the number of history queries and L_h is the padded length. More details are presented in Appendix A.

4.2 Contextual Information from Session Graphs

After we obtain the history query representations U_h, the next step is to refine them. Such refinement is necessary because the Transformer encoder considers the history queries separately, such that their interactions are not taken into account. However, since each search depends on its previous searches in the same session, modeling cross-query relations are imperative for determining the user's purchase

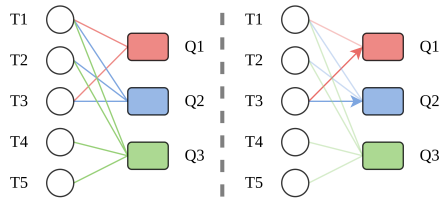


Figure 3: Left: Illustration of a session graph, where "*T*" stands for tokens and "*Q*" stands for queries. Right: One-step update based on the session graph.

intent. To this end, we use a graph attention mechanism (Velickovic et al., 2018; Wang et al., 2020) to capture contextual information from U_h.

4.2.1 Session Graph Construction

First we specify how to build a graph for each session, which we call the session graph. Suppose we have a session that contains three history queries:

$$\begin{aligned} Q_1 &: \{\textbf{Search query} : T_1, T_3\}, \\ Q_2 &: \{\textbf{Search query} : T_1, T_2, T_3\}, \\ Q_3 &: \{\textbf{Search query} : T_1, T_2, T_4, T_5\}, \end{aligned} \tag{1}$$

where Q_1, Q_2, Q_3 are the three queries, and $T_1, \cdots, T_5$ are the five tokens that appear in the three queries. Recall Section 3 for the problem setup. Figure 3 (left) illustrates the session graph.

4.2.2 Node Representations

The next step is to refine the node representations. Each of the nodes in the session graph has its own representation. The token representations are simply the corresponding representations of the tokens, extracted from the token embedding matrix. The query representations are the representations of the <boq> token in each padded history query, i.e., the representation of the Q_1 query node in Figure 3 is found by $U_h[0, 0, :] \in \mathbb{R}^d$. Denote $\mathcal{G}_q = \{q_i\}_{i=1}^{N_q}$ and $\mathcal{G}_t = \{t_i\}_{i=1}^{N_t}$ the sets of representations for the query and token nodes, respectively. Here N_q is the number of query nodes and N_t is the number of token nodes. Note that all the node representations have the same size, i.e., $q_i, t_i \in \mathbb{R}^d$.

4.2.3 Update Node Representations

We use a multi-head graph attention mechanism to update the node representations. For simplicity, denote $N_g = N_q + N_t$ the number of distinct nodes in the session graph, and $\mathcal{G} = \mathcal{G}_q \cup \mathcal{G}_t = \{g_i\}_{i=1}^{N_g}$ the set of all the node representations.

With the above notations, a single-head graph attention mechanism is defined as

$$\begin{aligned} h_i &= g_i + \text{ELU}\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} W_v g_j\right), \\ \text{where } \alpha_{ij} &= \frac{\exp(z_{ij})}{\sum_{\ell \in \mathcal{N}_i} \exp(z_{i\ell})}, \\ z_{ij} &= \text{LeakyReLU}\left(W_a [W_q g_i; W_k g_j]\right). \end{aligned} \tag{2}$$

Here $\text{ELU}(x) = x \cdot 1\{x > 0\} + (\exp(x) - 1) \cdot 1\{x \leq 0\}$ is the exponential linear unit, $\mathcal{N}_i$ denotes the neighbor of the i-th node, and W_a, W_q, W_k, W_v are trainable weights. Note that a residual connection (He et al., 2016a) is added to the last equation in Eq. 2. This has proven to be an effective technique to prevent gradient vanishing, and hence, to stabilize training.

The session graph only induces attention between nodes that are connected. For example, in Figure 3 (right), the model updates Q_1 and Q_2 using T_3, while Q_3 is unchanged, i.e., $\mathcal{N}_{T_3} = \{Q_1, Q_2\}$. A multi-head graph attention mechanism is then defined as the concatenation of $[h_i^1, h_i^2, \cdots, h_i^K]$, where K is the number of heads, and each of the h_i is calculated via Eq. 2.

The token node representations and the query node representations are updated iteratively. First, we update the token representations ($\mathcal{G}_t$) using the query representations ($\mathcal{G}_q$), in order that the tokens acknowledge to which queries they belong. Then, $\mathcal{G}_q$ is re-computed using the updated version of $\mathcal{G}_t$, which essentially evaluates cross-query relations, using the token nodes as intermediaries. Note that the graph attention mechanism (GAT) used in each of the two steps are distinct, i.e., there are two different sets of weights $[W_a, W_q, W_k, W_v]$.

Eventually, we obtain the updated vectorized representations $\{h_i\}_{i=1}^{N_g}$ for all the nodes, and we treat them as the contextual information of the session.

We remark that the GAT mechanism explicitly models cross-query relations by associating query representations with word representations. Such an approach is fundamentally different from existing methods, where the relations are either ignored (e.g., conventional Transformer attention) or captured via recursion (e.g., RNN-based approaches).

4.3 Session Representations

Recall that we pass the source query through a Transformer encoder and obtain $H_s \in \mathbb{R}^{L_s \times d}$. The matrix H_s contains representations for all the tokens in the source query. We use that of the

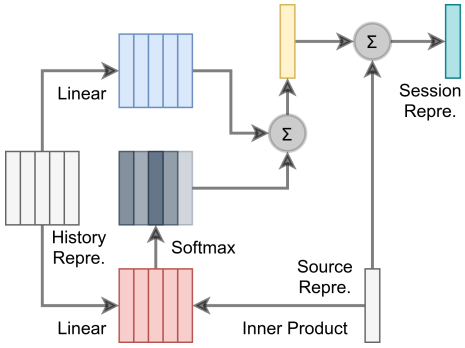


Figure 4: Aggregation network.

prepended <boq> token as the representation of the source query, which is denoted $h_s \in \mathbb{R}^d$. We adopt an aggregation network to extract useful information with respect to h_s from the contextual information $\{h_i\}_{i=1}^{N_h+N_t}$. The network employs an attention mechanism that determines to what extent each vector h_i contributes to the source query h_s. Figure 4 illustrates the architecture of the aggregation network. Concretely,

$$H_{\text{sess}} = H_s + \sum_{i=1}^{N_g} \alpha_i W_v h_i, \tag{3}$$

where $\alpha_i = \frac{\exp(z_i)}{\sum_{j=1}^{N_g} \exp(z_j)}$, $z_i = (W_k h_i)^\top h_s$, W_k and W_v are trainable weights. The summation in the last equation in Eq. 3 is conducted row-wise, wherein $H_{\text{sess}}, H_s \in \mathbb{R}^{L_s \times d}$, and $v \in \mathbb{R}^d$. The matrix H_{sess} serves as the representation of the session. Intuitively, by incorporating the aggregation network, we can filter out redundant information from the session history and only keep the ones pertinent to the source query.

After the Transformer encoder, the graph attention mechanism, and the aggregation network, we obtain H_{sess}, the session representation that contains information on both the source query and its history searches. Subsequently, H_{sess} is fed into the Transformer decoder to generate rewritten query candidates. The algorithm is detailed in Algorithm 1 in Appendix D.

5 Experiments

We conduct experiments on some in-house data. We implement two methods with different model architectures: ***Transformer+Aggregation+Graph*** and ***BART+Aggregation+Graph***. The first one is constructed in the previous section, and the second one employs a fine-tuning approach instead of training-from-scratch. The training details are deferred to Appendix B.3. More experimental results are shown in Appendix C.

5.1 Baselines

For baselines with pre-training, we use MeshBART (Chen and Lee, 2020) and BART (Lewis et al., 2020). For baselines without pre-training, we use LQRW (He et al., 2016b), HRED (Sordoni et al., 2015) and MeshTransformer (Chen and Lee, 2020) (a variant of MeshBART where we train the model from scratch). We also compare our algorithm with two model variants: *Transformer+Aggregation* and *BART+Aggregation*, where we use the aggregation network but not the GAT mechanism. Please refer to Appendix B.1 for details.

5.2 Evaluation Metrics

We use BLEU, MRR (Mean Reciprocal Rank), HIT@1, and HIT@16 to evaluate the query rewriting models. For all metrics except BLEU, we report the gains over the the results calculated by using only source queries. We remark that MRR, HIT@1, and HIT@16 (the percentage that the actual product is ranked within the first 16 products i.e., the first page when we search the rewritten query) are more important than BLEU, because MRR and HIT are directly linked to user experience. Please refer to Appendix B.2 for details about these metrics.

5.3 Experimental Results

Table 1 summarizes experimental results. Recall that in our formulation, we rewrite a source query to a target query. The "target query" entry in Table 1 is the performance gain of the ground truth target query, i.e., this entry signifies upper bounds of performance gain that any model can achieve.

We can see that the attention-based models (i.e., BART, MeshBART, Transformer and MeshTransformer) outperforms the recurrent neural network-based approach (i.e., LQRW and HRED). This is because RNNs suffer from forgetting and training issues. In contrast, Transformer-based models use the attention mechanism instead of recursion to capture dependencies, which has proven to be more effective. Moreover, by aggregating history searches, *BART+Aggregation* and *Transformer+Aggregation* consistently outperform their vanilla alternatives. Essentially performance of these two methods indicate that integrating history queries into training is critical. The performance is further enhanced by incorporating the session graphs. Specifically, *Transformer+Aggregation+Graph* achieves the best performance under almost all the metrics. Notice that the HIT@16 metric gain improves from +15.9 to

Table 1: Experimental results. The results of MRR, HIT@1, and HIT@16 are shown as gain over the source query. The best results are shown in **bold**.

Number of candidates	#Candidates=5			#Candidates=10			BLEU
Metric	MRR	HIT@1	HIT@16	MRR	HIT@1	HIT@16	
Source Query	0	0	0	0	0	0	—
Target Query	+16.1	+10.6	+29.0	+16.1	+10.6	+29.0	—
Baseline methods							
LQRW	+3.5	+2.5	+6.4	+6.8	+4.9	+12.6	29.4
HRED	+4.7	+3.2	+8.4	+8.1	+5.7	+14.2	25.7
BART	+4.6	+3.1	+8.2	+8.2	+5.5	+14.8	30.9
Transformer	+4.3	+2.6	+9.2	+8.5	+5.6	+15.9	25.3
MeshBART	+5.0	+3.8	+8.7	+8.3	+5.8	+14.3	31.7
MeshTransformer	+4.0	+2.7	+8.4	+8.3	+5.6	+15.7	25.9
Our methods							
BART+Aggregation	+6.3	+3.9	+10.9	+9.7	+6.4	+17.1	31.9
Transformer+Aggregation	+5.2	+2.9	+10.8	+10.2	+7.0	+17.3	27.2
BART+Aggregation+Graph	**+6.9**	**+4.6**	+11.8	+10.5	+7.5	+17.6	32.9
Transformer+Aggregation+Graph	+6.6	**+4.6**	**+12.0**	**+11.6**	**+8.3**	**+20.1**	28.2

+20.1 when employing both the aggregation network and the session graph formulation for the Transformer-based models. We highlight that the graph attention mechanism can directly captures cross-query relations, which is implausible for all the baselines. We can see that this property indeed contributes to model performance, i.e., HIT@16 increases from +17.3 to +20.1 when we equip *Transformer+Aggregation* with the GAT mechanism.

Notice that BLEU is not a definitive metric. For example, the MRR and HIT metrics of HRED are consistently higher than those of LQRW, even though the BLEU score of the former is significantly lower than the latter. Also, compared with Transformer-based models, the BLEU score is consistently higher when using the BART model as the backbone. This is because a pre-trained language model contains more semantic information. However, the MRR and HIT metrics of the BART-based models are slightly worse than those of the Transformer-based models.

However, the BLEU score is comparable for models with the same backbone. For example, for Transformer vs. *Transformer+Aggregation* vs. *Transformer+Aggregation+Graph*, the BLEU scores are 25.3 vs. 27.2 vs. 28.2. Such a tendency coincides with the online metrics. We observe the same results from BART-based models.

5.4 Online A/B Test

We conduct online A/B experiments on a large-scale e-commerce shopping platform with our query rewriting models. For a given search query within a session, we generate one reformulated query using the proposed model, and we feed both the original query and rewritten query into the search system. Experiments are conducted over five days, during which our system processed over 30 million sessions. The proposed method improves business metrics in terms of revenue; and also significantly decreases the number of reformulated searches. This indicates that the rewritten queries better meet customers' shopping intent since customers are able to find their desired products with less number of searches.

Table 2: Two examples of context-aware query rewriting with and without context.

Example 1 History	dodge led sign; dodge banners; mopar banner; mopar poster
Source	dodger posters
Target	dodge posters
Rewritten w/o context	dodger flag
Rewritten w/ context	dodge poster
Example 2 History	samsung galaxy case; samsung galaxy a11 case; samsung a11 case
Source	samsung galaxy a7
Target	samsung galaxy a7 case
Rewritten w/o context	samsung galaxy a7 charger
Rewritten w/ context	samsung galaxy a7 case

5.5 Case Studies

◇ Advantages of leveraging history information
Two examples are shown in Table 2. The first example is error correction. In the example, the customer wishes to purchase dodge (a car brand) posters, but she mistakenly searches for dodger (a baseball team) posters. Without history information, it is impossible to determine the customer's true intent.

Table 3: Two examples of generated queries and their associated likelihood.

Type	Query	Likelihood	Query	Likelihood
History	iphone 11 pro case pokemon; iphone 11 pro case eevee; iphone 11 pro case hetalia; iphone 11 pro case sailor moon	—	colorado 2005 tail lights; colorado 2005 door colorado 2005 accessories	—
Source	iphone 11 pro case snow leopard	—	colorado headlights	—
Target	iphone 11 pro case tiger	—	colorado 2005 headlights	—
Rewritten	iphone 11 pro case disney	0.497	2005 colorado headlights	0.566
	iphone 11 pro case sailor moon	0.492	colorado headlights 2005	0.458
	iphone 11 pro case harry potter	0.445	colorado headlights led	0.357
	iphone 11 pro case	0.440	colorado headlights assembly	0.301
	iphone 11 pro case cute	0.419	colorado tail lights	0.289
	iphone 11 pro case leopard	0.391	colorado headlights housing	0.237
	iphone 11 pro case clear	0.379	colorado led headlights	0.234
	iphone 11 pro case disney princess	0.372	2004 colorado headlights	0.230
	iphone 11 pro case pink	0.364	colorado 2004 headlights	0.214
	iphone 11 pro case totoro	0.353	colorado headlights 2004	0.208

However, by looking at session histories, we find that all the previous searches are related to automobiles (e.g., dodge and mopar), and therefore the query should be rewritten to "dodge posters". Our model successfully captures this pattern. Notice that the rewritten query without leveraging context does not match the user's intent.

The second example is keyword refinement. In the example, by looking at the history searches, it is obvious that the customer wishes to find phone cases, instead of phones. However, this intent is impossible to capture by using only the source query. Our model automatically adds the keyword "case" to the source query and matches the target query. On the other hand, without the context information, the rewritten result is not satisfactory.

◇ **Diversity of query generation** Table 3 demonstrates two examples. In the first example (the left three columns), notice that our model can grep information from history queries, e.g., "iphone 11 case sailor moon", and can delete keywords that are deemed insignificant or too restrictive, e.g., "iphone 11 case leopard" instead of "snow leopard". Also, our model can effectively capture domain information. For example, some of the history query keywords (e.g., pokemon, eevee) are often described as "cute", and our model recommends this keyword. All the history keywords are from Japanese anime series, therefore our model suggests another popular character, "totoro". Additionally, the "disney" and "disney princess" keywords are generated based on the interest to virtual characters. Finally, notice that the likelihood of all the suggested queries is similar, which means our model cannot single out a significantly better query than the others. Therefore our model generated a diverse group of queries.

In the second example (the right two columns), the generated query successfully matches the target query. Note that the top two generated queries have high likelihood, and the likelihood decreases drastically as the suggested queries become more and more implausible. In this example, the first query is 172% more likely than the tenth query, whereas this number is only 41% in the previous example. This suggests that our model can differentiate between good quality suggestions and poor quality alternatives.

6 Conclusion and Discussion

We propose an end-to-end context-aware query rewriting model that can efficiently leverage user's history behavior. Our model infers a user's purchase intent by modeling her history searches as a graph, on which a graph attention mechanism is applied to generate informative session representations. The representations are subsequently decoded into rewritten queries. Our proposed session graph can be extended to incorporate more information. Here, we present a bipartite graph, which contains words and queries. Additional components can be added as extra layers. For example, we can add product information such as categories to the session graph, which will create 3-partite session graphs (word, query and product).

References

Ricardo Baeza-Yates and Alessandro Tiberi. 2007. Extracting semantic relations from query logs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 76–85.

Bodo Billerbeck, Falk Scholer, Hugh E Williams, and Justin Zobel. 2003. Query expansion using associated queries. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 2–9.

Huanhuan Cao, Daxin Jiang, Jian Pei, Qi He, Zhen Liao, Enhong Chen, and Hang Li. 2008. Context-aware query suggestion by mining click-through and session data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 875–883.

Ruey-Cheng Chen and Chia-Jung Lee. 2020. Incorporating behavioral hypotheses for query generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3105–3110, Online. Association for Computational Linguistics.

Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. 2002. Probabilistic query expansion using query logs. In *Proceedings of the Eleventh International World Wide Web Conference, WWW 2002, May 7-11, 2002, Honolulu, Hawaii, USA*, pages 325–332. ACM.

Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. 2003. Query expansion by mining user logs. *IEEE Transactions on knowledge and data engineering*, 15(4):829–839.

Mostafa Dehghani, Sascha Rothe, Enrique Alfonseca, and Pascal Fleury. 2017. Learning to attend, copy, and generate for session-based query suggestion. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pages 1747–1756. ACM.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, Fabrizio Silvestri, and Narayan Bhamidipati. 2015. Context- and content-aware embeddings for query rewriting in sponsored search. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 383–392. ACM.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016a. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

Yunlong He, Jiliang Tang, Hua Ouyang, Changsung Kang, Dawei Yin, and Yi Chang. 2016b. Learning to rewrite queries. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 1443–1452. ACM.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Chien-Kang Huang, Lee-Feng Chien, and Yen-Jen Oyang. 2003. Relevant term suggestion in interactive web search based on contextual information in query session logs. *Journal of the American Society for Information Science and Technology*, 54(7):638–649.

Jeff Huang and Efthimis N Efthimiadis. 2009. Analyzing and evaluating query reformulation strategies in web search logs. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 77–86.

Jyun-Yu Jiang and Wei Wang. 2018. RIN: reformulation inference network for context-aware query suggestion. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, pages 197–206. ACM.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

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

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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

Stefan Riezler, Alexander Vasserman, Ioannis Tsochantaridis, Vibhu Mittal, and Yi Liu. 2007. Statistical machine translation for query expansion in answer retrieval. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 464–471, Prague, Czech Republic. Association for Computational Linguistics.

Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. 2015. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*, pages 553–562. ACM.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Danqing Wang, Pengfei Liu, Yining Zheng, Xipeng Qiu, and Xuanjing Huang. 2020. Heterogeneous graph neural networks for extractive document summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6209–6219, Online. Association for Computational Linguistics.

Xuanhui Wang and ChengXiang Zhai. 2007. Learn from web search logs to organize search results. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 87–94.

Xuanhui Wang and ChengXiang Zhai. 2008. Mining term association patterns from search logs for effective query reformulation. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 479–488.

Rong Xiao, Jianhui Ji, Baoliang Cui, Haihong Tang, Wenwu Ou, Yanghua Xiao, Jiwei Tan, and Xuan Ju. 2019. Weakly supervised co-training of query rewriting andsemantic matching for e-commerce. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*, pages 402–410. ACM.

Yatao Yang, Jun Tan, Hongbo Deng, Zibin Zheng, Yutong Lu, and Xiangke Liao. 2019. An active and deep semantic matching framework for query rewrite in e-commercial search engine. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 309–318. ACM.

A Transformer Encoder Details

For a given source query, we first pad it with a <boq> (begin-of-query) token. Then, we pass the padded query through a token embedding layer and a position embedding layer, and we obtain $Y_s \in \mathbb{R}^{L_s \times d}$. Here L_s is the length of the padded source query, and d is the embedding dimension. Note that the position embedding can either be a sinusoidal function or a learned matrix.

After the initial embedding layers, we pass Y_s through the self-attention module. Specifically, we compute attention output S by

$$S = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}V\right), \quad \text{where } Q = Y_s W_q,\ K = Y_s W_k,\ V = Y_s W_v. \tag{4}$$

Here $W_q, W_k \in \mathbb{R}^{d \times d_K}$, $W_v \in \mathbb{R}^{d \times d_V}$ are learnable weights. In practice we use multi-head self-attention to increase model flexibility. To facilitate this, different attention outputs $S_1, \cdots, S_H$ are computed using different sets of weights $\{W_q^h, W_k^h, W_v^h\}_{h=1}^H$. The final attention output is

$$S = [S_1, S_2, \cdots, S_H]\, W_o, \tag{5}$$

where $W_o \in \mathbb{R}^{Hd_V \times d}$ is a learnable aggregation matrix. The attention output is then fed through a position-wise feed-forward neural network to generate encoded representation $H_s \in \mathbb{R}^{L_s \times d}$ for the source query:

$$H_s = \text{ReLU}\left(S\, W^1_{\text{FFN}} + b^1\right) W^2_{\text{FFN}} + b^2. \tag{6}$$

Here $\{W^1_{\text{FFN}}, W^2_{\text{FFN}}, b^1, b^2\}$ are weights of the neural network. Equations 4, 5, and 6 constitute as an encoder block. In practice we stack multiple encoder blocks to build the Transformer encoder, as demonstrated in Figure 2.

For the history queries in this session, we also pad them with <boq> tokens. Suppose that we have N_h padded history queries (recall a session contains multiple history queries), and their respective length is denoted by $L_h^1, \cdots, L_h^{N_h}$. We pad the history queries to the same length, and we obtain the history query matrix $X_h \in \mathbb{R}^{N_h \times L_h}$, where $L_h = \max\{L_h^1, \cdots, L_h^{N_h}\}$. Then, following the same procedures as encoding the source query, we pass X_h through the embedding layers and the encoder blocks, after which we obtain the history query representations $U_h \in \mathbb{R}^{N_h \times L_h \times d}$.

B Experiments Details

B.1 Baselines

The baselines are split into two groups: without pre-training and with pre-training. For the w/o pre-training group, we build the following models:

◇ ***Learning to Rewrite Queries (LQRW)*** (He et al., 2016b) is one of the first methods that applies deep learning techniques to query rewriting. Specifically, the LQRW model combines a sequence-to-sequence LSTM (Hochreiter and Schmidhuber, 1997; Sutskever et al., 2014) model with statistical machine translation (Riezler and Liu, 2010) techniques to generate queries. The candidates are subsequently ranked using hand-crafted feature functions.

◇ ***Hierarchical Recurrent Encoder-Decoder (HRED)*** (Sordoni et al., 2015) employs a hierarchical recurrent neural network for generative query suggestion. The model is a step forward from its predecessors in that HERD is sensitive to the order of queries and the method is able to suggest rare and long-tail queries.

◇ ***Transformer*** (Vaswani et al., 2017) has achieved superior performance in various sequence-to-sequence (seq2seq) learning tasks. To adopt Transformer to query rewriting, we treat the source query as the source-side input, and the target query as the target-side input. Then we train a model using only these constructed inputs, similar to machine translation. Note that this setting resembles most of the existing works. We adopt the Transformer-base architecture, which contains about 72M parameters.

◇ ***MeshTransformer*** (Chen and Lee, 2020) is a variant of MeshBART, where the pre-trained BART module is replaced by a Transformer and the model is trained from scratch. The method concatenates history queries to the source query in order to integrate contextual information. See the MeshBART method below for details.

◇ ***Transformer+Aggregation*** is the model where we use the aggregation network to encode history search queries, i.e., without the graph attention mechanism. Specifically, we first obtain the representations of the source query and the history queries from the Transformer encoder. Then, we extract information related to the source query from the history representations using an aggregation network. Such information is added to the source representation, and we follow a standard decoding

procedure using these two factors. See Section 4.3 for details.

The second group of methods adopt pre-trained language models for query rewriting.

◇ ***BART*** (Lewis et al., 2020) is a pre-trained seq2seq model. We adopt this particular model instead of, for example, BERT (Devlin et al., 2019) or GPT-2 (Radford et al., 2019), because we treat query rewriting as a seq2seq task. And the aforementioned architectures have either the Transformer encoder (e.g., BERT) or the Transformer decoder (e.g., GPT-2), but not both. In our experiments, BART is fine-tuned in a setting similar to training the Transformer model. We adopt the BART-base architecture in all the experiments, which contains about 140M parameters.

◇ ***MeshBART*** (Chen and Lee, 2020) is a BART-based model that first concatenates the history queries to the source query, and then feeds the concatenated input to a pre-trained BART model for query generation. Note that the original method requires click information. We remove this component as the proposed method do not need such data.

◇ ***BART+Aggregation*** is similar to *Transformer+Aggregation*, except we replace the Transformer backbone with the pre-trained seq2seq BART model.

B.2 Evaluation Metrics

We use the BLEU score (Post, 2018) as an evaluation metric. This metric is constantly used to evaluate the quality of translation. We adopt it here because similar to machine translation, we formulate query rewriting as a seq2seq learning task. The correlation between the rewritten query and the target query reflects the model's ability to capture the user's purchase intent.

The MRR metric describes the accuracy of the rewritten queries. For each source query in the test set, we generate 10 candidate queries $r_1, \cdots, r_{10}$. Then we search each of these candidates using our production search engine, and we obtain the returned products, of which we only keep the top 32. Recap that we know the actual product that the customer purchased. The next step is to calculate the reciprocal of the actual product's rank for each of $r_1, \cdots, r_{10}$. For example, suppose for r_1, the actual purchased product is the second within the 32 returned products, then the score for r_1 is $\text{score}_1 = 1/2 = 0.5$. The score of the rewritten queries $r_1, \cdots, r_{10}$ is then defined as $\max\{\text{score}_i\}_{i=1}^{10}$. Finally, the score for the query rewriting model is the average over all the source query scores.

We also use HIT@1 and HIT@16 as evaluation metrics. The HIT@16 metric is the percentage that the actual product is ranked within the first 16 products (the first page) when we search the rewritten query. And the HIT@1 metric is similarly defined.

B.3 Training Details

We use the *Fairseq* (Ott et al., 2019) code-base with *PyTorch* (Paszke et al., 2019) as the back-end to implement all the methods. All the experiments are conducted using 8 NVIDIA V100 (32GB) GPUs.

For training a Transformer model from scratch, we adopt the Transformer-base (Vaswani et al., 2017) architecture. We use Adam (Kingma and Ba, 2015) as the optimizer, and the learning rate is chosen from $\{3 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}$. We use 4 heads for the multi-head graph attention mechanism, where the head dimension is set to be 128 (note that the Transformer-base architecture has embedding dimension 512).

For fine-tuning a BART model, we adopt the BART-base (Lewis et al., 2020) architecture. We use AdamW (Loshchilov and Hutter, 2019) as the optimizer, and the learning rate is chosen from $\{3 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}\}$. Similar to the training from scratch scheme, we adopt 4 heads, each with dimension 192, for the graph attention mechanism.

For both training-from-scratch and fine-tuning, please refer to[1] Ott et al. (2019) for more details such as pre-processing steps and other hyper-parameters.

C More Experimental Results

C.1 Analysis

◇ **BART vs. Transformer** Even though BART contains twice the number of parameters compared with Transformer (140M vs. 70M), models fine-tuned on BART yield lower MRR and HIT metrics (Table 1). One reason is that publicly available pre-trained models are pre-trained on natural language corpus, but queries are usually short and have distinct structures. This raises doubts on whether cur-

[1] https://github.com/pytorch/fairseq/blob/master/examples/translation/README.md

rent pre-trained models are suitable for the query domain. Indeed, the rich semantic information enables a much better BLEU score (32.9 vs. 28.2), but the MRR and HIT metrics suggest the fine-tuned models' unsatisfactory performance.

Another reason is that in a conventional fine-tuning task, a task-specific head is appended to the pre-trained model, and the head usually contains only a small number of parameters. But in the query rewriting task, both the aggregation network and the graph attention mechanism contain a significant amount of parameters (about 10% of BART). This is problematic because in fine-tuning, the learning rate is usually small since nearly all the weights are supposed to be meaningful and should not change much. Yet, in our case, we need to properly train a large amount of randomly initialized parameters. Moreover, the aggregation network and the GAT are added inside the pre-trained model (more specifically, they are added to the BART encoder) instead of appended after BART. Essentially this nullifies the pre-trained parameters on the decoder side, imposing additional challenges to the fine-tuning task. Nevertheless, the *BART+Aggregation* model still outperforms the vanilla BART model, and the performance is further improved by adding the GAT (i.e., *BART+Aggregation+Graph*).

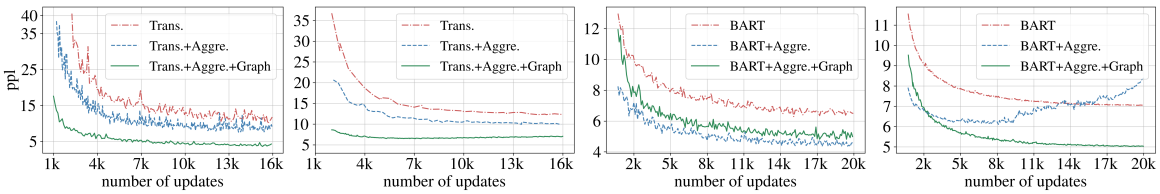
◇ **Training from scratch vs. fine-tuning** Figure 5 plots the training and validation perplexity (ppl) of the training-from-scratch approach and the fine-tuning approach. From Figure 5a and Figure 5b, we can see that by employing the aggregation network, *Transformer+Aggregation* fits the data better and exhibits enhanced generalization. The training and validation ppls are further significantly improved by incorporating the graph attention mechanism, i.e., by using *Transformer+Aggregation+Graph*, we achieve even better performance.

Notice that in Figure 5c, *BART+Aggregation* outperforms *BART+Aggregation+Graph* in terms of training ppl, which is different from the training-from-scratch approach. As indicated by Figure 5d, *BART+Aggregation* shows clear sign of over-fitting. This is because even though pre-trained language models contain rich semantic information, much of it is considered "noisy" for query rewriting. Thus feature enhancement initiated by the graph attention mechanism is needed.

◇ **Model size vs. performance** Figure 6 illustrates the relation between model size and performance, where we decrease the embedding dimension (correspondingly the FFNs' hidden dimensions) and the number of layers. We can see that even with $1/8$ of the parameters, model performance does not decrease much. Moreover, our model is more than 20% smaller than a BERT-base model (85M vs. 110M), rendering online deployment more than possible.

◇ **Query length vs. performance** Figure 7 demonstrates model performance regarding length of the instant query. We can see that the BLEU score gradually decreases when the length increases. This is because long queries are often very specific (e.g., down to specific models or makes), making the rewriting task harder.

D Detailed Algorithm



(a) Transformer (train). (b) Transformer (valid). (c) BART (train). (d) BART (valid).

Figure 5: Training and validation perplexity using Transformer and BART as backbone.

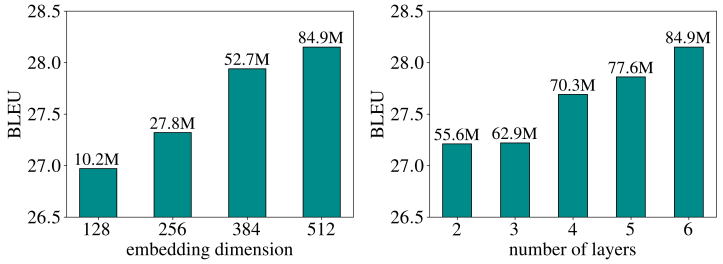


Figure 6: Model performance (in BLEU scores) vs. model size. The model size (in millions of parameters) are shown above the bars.

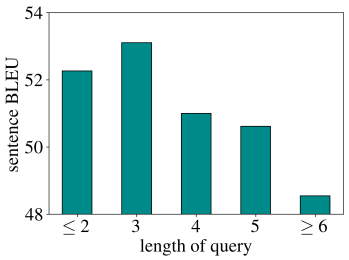


Figure 7: Query length vs. rewriting quality.

Algorithm 1: Context-aware query rewriting.

Input: $\mathcal{D}$: dataset containing sessions; Initial parameters for the Transformer encoder and the Transformer decoder; Initial parameters for two graph attention mechanism (Eq. 2): $\mathrm{GAT}_{\mathrm{t}\to\mathrm{q}}$, $\mathrm{GAT}_{\mathrm{q}\to\mathrm{t}}$; Initial parameters for the aggregation network (Eq. 3); K: the number of updates on the session graph; N: the number of rewritten queries for each session.

Output: A list that contains N generated queries for each session in the dataset.

Rewritten results: $\mathrm{rewritten} = \{\}$;

for *each session in* $\mathcal{D}$ **do**

`/* Encode input data. */`

Compute source representation H_s and history representation U_h using the Transformer encoder;

`/* Apply graph attention. */`

Obtain initial representations $\mathcal{G}_t^0, \mathcal{G}_q^0$;

for $k = 1 \cdots K$ **do**

$\mathcal{G}_t^k = \mathrm{GAT}_{\mathrm{q}\to\mathrm{t}}(\mathcal{G}_q^{k-1}, \mathcal{G}_t^{k-1})$;

$\mathcal{G}_q^k = \mathrm{GAT}_{\mathrm{t}\to\mathrm{q}}(\mathcal{G}_t^k, \mathcal{G}_q^{k-1/2})$;

end

Set history representation $\{h_i\}_{i=1}^{N_t+N_h} = \mathcal{G}_t^K \cup \mathcal{G}_h^K$;

`/* Apply aggregation network. */`

Compute session representation H_{sess} from H_s and $\{h_i\}_{i=1}^{N_t+N_h}$ using Eq. 3;

`/* Generate rewritten queries. */`

Generate N rewritten queries $\{q_i\}_{i=1}^{N}$ using the Transformer decoder and a beam search procedure;

$\mathrm{rewritten} \leftarrow \mathrm{rewritten} \cup \{q_i\}_{i=1}^{N}$;

end

Output: The rewritten queries.