Multi-objective Neural Retrieval for Query AutoComplete

Rohit Patki
Amazon Search
Palo Alto, USA

Sravan Bodapati
Amazon Search
Palo Alto, USA

Christopher Potts
Stanford University
Stanford, USA

ABSTRACT

Query Autocomplete (QAC) systems predict the best query suggestions based on customer typed prefix and other contextual signals. Conventional techniques employ the Most Popular Completion (MPC) method, where query suggestions that are popular and begin with the prefix (prefix aware) are retrieved from a pre-computed index. To account for contextual signals like past search activity of the user in the session, QAC systems incorporate a re-ranking step on top of retrieved candidates. However, this is sub-optimal as the retrieved candidates do not necessarily contain any session relevant query suggestions. We propose an efficient way to retrieve session relevant, prefix aware and popular query suggestions at the same time. We present a vector transformation technique to combine different objective representations into one which is then used to search in a pre-computed vector index at inference time. We show that our method improves recall@100 over MPC and other baselines by 13% to 15% on one e-commerce dataset and AOL query logs without incurring significant latency.

ACM Reference Format:
Rohit Patki, Sravan Bodapati, and Christopher Potts. 2024. Multi-objective Neural Retrieval for Query AutoComplete. In *Proceedings of 3rd International Workshop on Interactive and Scalable Information Retrieval methods for eCommerce (ISIR-eCom) 2024 @ WSDM-2024, https://isir-ecom.github.io/ (ISIR-eCom 2024 @ WSDM-2024).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Query Autocomplete (QAC) systems recommend query completions based on the partial query (prefix) typed by the customer in the search box.The goal of QAC is not only to suggest user's intended query after minimal input keystrokes, but also to rank the user's intended query highly while also recommending queries that lead to the best search engine results. Recent advancements in natural language processing [13] have enabled advanced semantic understanding of text beyond just lexical features. These methods have improved understanding of user's intent even from partially typed queries in QAC. The primary focus of our work is to apply QAC in e-commerce search engines, aiming to suggest queries that lead customers to the most relevant product result page.

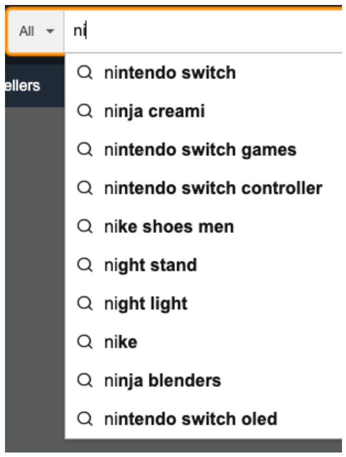

Figure 1: Example of QAC

A QAC system, like any Information Retrieval (IR) system, typically involves two steps: candidate generation and ranking. Candidate generation produces a list of K relevant query suggestions, often using a method like MPC [4] which retrieves most popular query completions for a prefix from a cached database. The ranking step employs more sophisticated models like [14], [3] and considers features such as popularity, relevance to the customer's past activity, and purchase conversion rate, to create an ordered list of top suggestions for display.

The challenges with QAC involve understanding customer intent from partially typed queries and utilizing additional information available to the system, such as the customer's past activities like queries, product clicks, and purchases. Since customers make multiple query searches and browse multiple products before making a purchase, it is obvious that past session activity is a good predictor of next query search. Many previous methods like [6], [2], [12] and [8] focus on using session context and personalized features during the ranking phase. We advocate the importance of incorporating session context during the candidate generation phase itself, adhering to the prefix typed by the customer. The impact from absence of session relevant candidates would be more pronounced for shorter prefixes (eg. "n" or "ni"), as MPC would narrow down huge number of possible candidates to a very small number, thereby reducing the probability of retrieving session relevant candidates in the first step. For longer prefixes(e.g. "nike sho"), its likely MPC might contain session relevant candidates as the number of possible queries isn't huge, but this wouldn't align with the goals of QAC to reduce customer's typing efforts.

An alternative way to increase session relevant candidates in the candidate generation step is to retrieve more candidates, but this inevitably increases the latency of the ranking step. To address this challenge, we propose a multi-objective neural retrieval method (MONR). We combine different attributes of a query, such as semantic representation using BERT-based models [5], spelling of the query and popularity in a single vector which is then indexed offline using libraries built for fast similarity search of dense vectors. During the candidate generation step, we create a similarly sized input vector based on user session context and prefix. The input vector is then used to perform maximum inner product search in the pre-computed vector index to get top K candidates. These candidates are then sent to the ranker. Our method retrieves session relevant, popular and prefix aware candidates efficiently and from a single pre-computed index. Through this work, we make the following contributions:

- **Incorporation of Session Context into candidate generation:** This enhances the quality of queries fed into the ranking step, thereby improving the recall in the process.
- **Character Encoding of Prefix:** We encode the characters of a prefix ("n","i","k" for "nik" prefix) as a vector such that it has maximum inner product with any query that begins with the prefix. This aids in retrieving relevant query candidates for the customer's prefix.
- **Multi-objective Similarity Search:** We introduce a vector transformation technique that combines various query attributes, enabling multi-objective similarity search. Additionally, this technique can be applied for filtering in vector similarity indexes.

2 RELATED WORK

Neural Retrieval Models: Retrieval of semantically similar documents in response to a given input text has been extensively studied in IR. BM25 [1], typically considered a sparse retrieval method, is a strong baseline that retrieves documents based on tokens in the input text. Recent research shows enhancements over BM25 through dense retrieval methods like Dense Passage Retriever(DPR) [9]. These methods transform input text into dense vectors and conduct similarity searches based on cosine similarity within a pre-computed vector index of dense document vectors. There are many fast and efficient libraries available to perform vector search such as FAISS [7] which allow quick semantic retrieval of documents using dense representations of input. ColBERT [10], an end-to-end method that integrates retrieval and ranking in a unified step, demonstrates that late interaction over dense token representations in the input and documents, as opposed to cosine similarity over single dense representations, can capture relevance more effectively. ColBERT serves as a favorable middle ground between BM25-style sparse retrieval and DPR techniques. Our method can be seamlessly integrated with any of these neural retrieval techniques, incorporating not only the semantic similarity between session context and the predicted query but also essential features like query popularity and prefix awareness for the query suggestions.

3 MULTI OBJECTIVE RETRIEVAL

For QAC, we have the following objectives:

- The suggestions are **relevant to the customer's activity in the session**. In this work, we use the customer's previous query within the last 5 minutes as the session context.
- The suggestions **respect the prefix** typed by the customer.
- **Popular** suggestions are retrieved before less popular suggestions.

The inputs to QAC are previous query and current prefix. We represent these together in a vector (input vector). The suggestion candidates are also represented as vectors (suggestion vectors). The suggestion vectors are indexed offline using FAISS [7] and during inference, we retrieve the top K suggestions for the given input vector by finding its K nearest neighbors in the vector index using maximum inner product search. Below, we explain the construction of the input and suggestion vectors, aiming to optimize the retrieval process for the above-mentioned objectives.

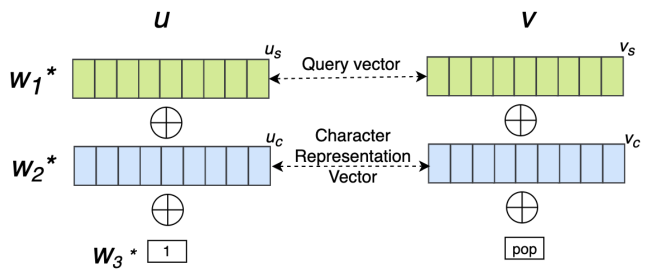

Figure 2: Vector Creation

3.1 Input and Suggestion Vectors

Both input (u) and suggestion vectors (v) are concatenation of three vectors as shown in 2: (1) Query vector, (2) Character Representation Vector, and (3) Popularity vector. Below we describe how each of the vectors are constructed.

3.1.1 Query vector. A user typically makes multiple query searches before making a purchase. Therefore QAC should suggest queries that are related to previous queries in the session. We propose to achieve this by using cosine similarity between previous query and next query candidates. We use any encoder model (e.g. BERT [5]) to compute vector representation for queries. For input vector, we encode the previous query of the user. For suggestion vector, we encode the suggestion query. Note that, the encoder model is usually fine-tuned on the application logs so that it can recognize associations between application related similar queries (e.g. "nike" and "adidas" are related in e-commerce context). In above figure, u_s is the previous query vector in input vector and v_s is the next query vector in suggestion vector. They are both normalized so that the dot product is between 0 and 1.

3.1.2 Character Representation Vector. QAC needs to respect the user prefix, as the users expect query suggestions to start with their prefix. We need to represent the prefix string as a vector that captures the order of characters such that the dot product with suggestion string vector is highest when suggestion starts with the prefix. We propose to represent the prefix and suggestion strings with a vector of dimension m where m is the maximum string length permitted (In this work, we pick a value of $m = 50$).

We map each letter in the prefix/suggestion string to a unique position in the vector. The value at that position is a function of the position of the letter in the string. The value decreases exponentially as position increases. For example, the value at the position corresponding to the n^{th} letter is e^{-n}. We exponentially decrease the values because it is more important to match the prefix with suggestion at starting positions (e.g., "pap" and "app" are not similar). In above figure, u_c is the prefix string vector and v_c is the suggestion string vector. Pseudo code to create the string vector can be found in the appendix. The prefix character representation vector is normalized by square of the norm to ensure that the dot product with suggestion string vector will at most be 1.

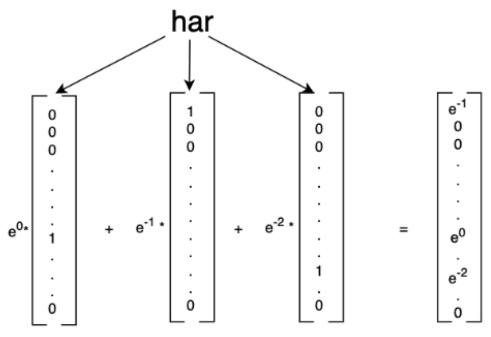

Figure 3: Character Representation Vector

3.1.3 Popularity vector. Popular queries by definition have higher probability of being searched by users. Therefore it is important for QAC to account for query popularity. For example, if the last query was "shoes" and the prefix is "n", suggestions "nike shoes" and "nike trail running shoes" are equally relevant to the session and the prefix but if "nike shoes" is 10 times more popular than "nike trail running shoes", then it should be retrieved with a higher score. The query vector dot product will likely have higher value for "nike shoes" by virtue of seeing it more often in the training data but the popularity of queries can change wildly depending on the day or season. A simple solution to this problem is to retrain the session vector encoder quite regularly, but it's impractical to keep retraining it. It is easier to achieve this by explicitly making use of the popularity score during retrieval. We achieve this by representing popularity as a one dimensional vector of normalized log popularity for suggestion vector. Using logarithm reduces the impact of outliers and normalizing the log value with maximum log value constrains the value to lie between 0 and 1. For popularity vector of the user input, we use a constant value of 1. As a result, the dot product yields the normalized log popularity.

3.2 Retrieval Score

We will compute multi objective retrieval score of a query suggestion for a given input as a linear combination of the above three objectives. The session relevance is cosine similarity between previous query in the session and the query suggestion candidate. The prefix aware score is dot product between the string vectors of prefix and query suggestion. The popularity score is normalized log popularity of query suggestion, or dot product between popularity vectors of input and suggestion. The weights on these scores can either be manually decided as hyper-parameters based on recall improvement or we can learn them with a logistic regression model described in section 4.3 below.

$$\begin{aligned} p(s|i) &\propto w_1 * u_s.v_s + w_2 * u_c.v_c + w_3 * 1.pop \\ &\propto u.v \end{aligned} \tag{1}$$

Therefore the multi-objective score is the dot product between the input and suggestion vectors. The task of the retrieval system now reduces to finding top K suggestion vectors with maximum inner product with a given input vector. Dealing with vector spaces and inner products allows us to easily leverage libraries like FAISS (Johnson et al., 2017) to place suggestion vectors in a vector index offline and find the top K nearest neighbors online. Thus, we have successfully transformed a multi objective retrieval problem into a vector search problem. As the weights on different objectives are only part of the input vector, they can modified as desired at inference time. For instance, you might prioritize more relevance and less popularity when dealing with long prefixes or a larger number of queries in the session context. Conversely, you may want to reduce prefix awareness for longer prefixes, especially in cases where there could be a spelling mistake.

Below we describe how these weights can be learned using our proposed training method.

4 PROPOSED SYSTEM

The retrieval method is summarized in the picture above. All the suggestions are vectorized offline using the query encoder, string encoder and popularity vector. The suggestion vectors are stored in a vector index (e.g. FAISS) primed for fast maximum inner product search (MIPS). When the customer types in the search box, the input is first vectorized using the query encoder and string encoder. This vector is used to perform top-K MIPS to get relevant suggestions for the customer. These K candidates can be further sent to a re-ranking step but that is out of scope for this paper. The goal of the retriever is to produce a list of highly relevant suggestions. We measure the quality of retriever by computing recall, that is, how often is the customer searched query present in the top-K list produced by the retriever. Below, we describe how each of the components of the system are trained. We first train the BERT-based query encoder and then train the linear weights on three objectives.

4.1 Query Encoder

The goal of query encoder is to produce vector embeddings such that customer searched query suggestion embedding has higher inner product with embedding of the last query in the session compared to all the other suggestions. We train this encoders with a bi-encoder architecture like in DPR but the weights are shared.

We use QAC logs to generate triples $<s, q^+, q^->$ where q^+ is the submitted query and q^- is a randomly sampled query not submitted by the customer. s is the previous query of the user. We train the query encoder with a pairwise cross entropy loss function. As a result, the encoder learns to generate similar embeddings for queries in the same session.

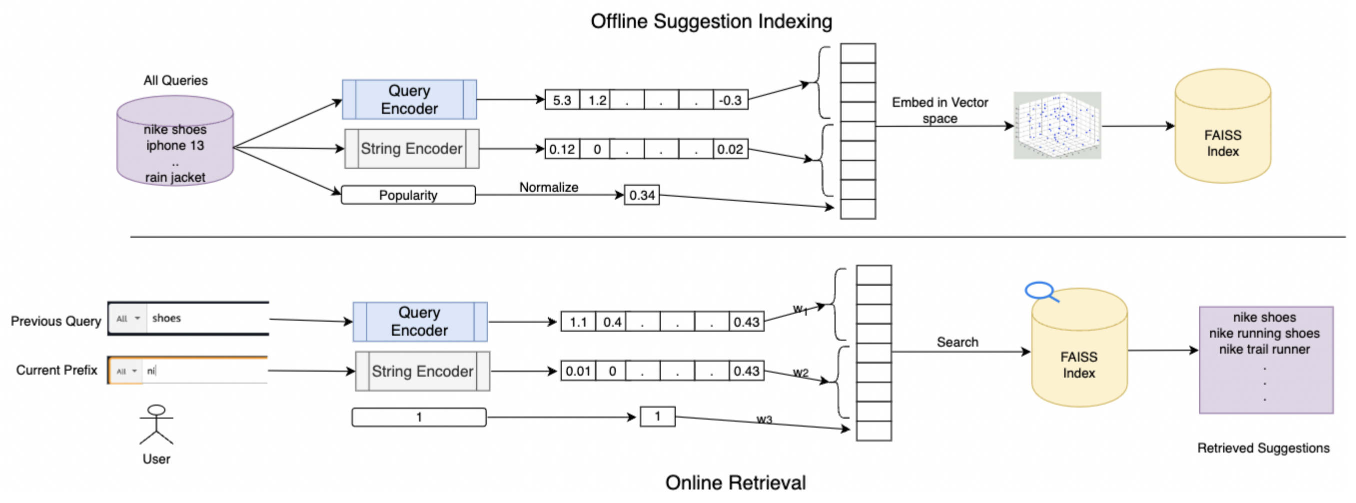

Figure 4: Proposed System

4.2 Linear Weights

The linear weights w_1, w_2 and w_3 are important to identify how important each of the objectives are to improving recall. We learn these weights with a logistic regression model. The positive samples are submitted queries and negative samples are sampled from semantically similar queries using query encoder but not submitted by the customer.

4.3 Offline Indexing

Once the training is completed, we index the concatenated vector representation of suggestions using FAISS. This index is loaded into memory in QAC service. As customer types a prefix, the query encoder computes the embedding for previous query if available and the character representation encoder generates prefix embedding. These two are each multiplied by w_1 and w_2 before being concatenated with w_3 valued vector of unit dimension. We search the loaded FAISS index with this vector and return top-K results.

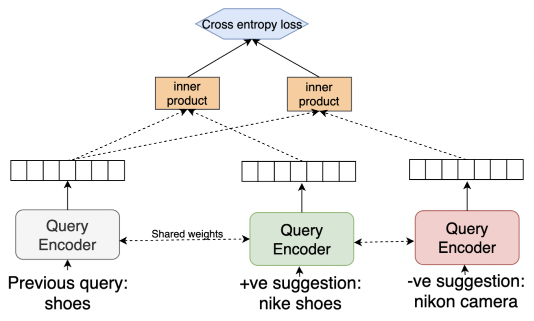

Figure 5: Encoder training

5 EXPERIMENTAL RESULTS

We now empirically test our method against 2 baselines using recall@k metric. We will use MPC as one baseline where top-k for a given <last query, prefix> combination would mean that we would retrieve the top-k most popular query suggestions. One intuitive way to incorporate session context into candidate generation is to retrieve top-$k/2$ semantically similar queries to *previous query* and combine it with top-$k/2$ MPC candidates. We use this as a second baseline. For our method (MONR), we retrieve top-k from FAISS vector index pre-computed from dense representations as mentioned in section 3. We ensure that the comparisons are made with same k. If the customer submitted query is present in the retrieved k query suggestions, we label its recall 1, 0 otherwise. We evaluate our method on two datasets: (1) proprietary e-commerce QAC dataset and, (2) public AOL query logs.

E-commerce QAC Dataset: The e-commerce QAC dataset we use consists of rows which contain past search of the customer, current prefix and submitted query. We aggregate four weeks of data logs to compute the popularity and train the query encoder. We employ the data from the 5th week to compute the linear weights, as outlined in section 4.3. We sample data in week 6 to report recall metrics at different ks.

Table 1: The recall results for E-commerce dataset at different ks. We report relative recall in comparison to recall at 10 for MPC baseline

Model	R@10	R@50	R@100
MPC (Baseline)	-	1.82	1.97
MPC + Semantic	0.86	1.8	1.92
MONR	**1.2**	**2.08**	**2.22**

AOL Query Logs: The dataset [11] has 16M queries submitted by 657K unique users sampled between 1 March, 2006 and 31 May, 2006 from AOL website. We aggregate the dataset such that each query is represented in one row. We consider the previous user query made within 5 minutes as session context. We used the queries submitted before 15 May 2006 for computing query popularity. We use pretrained *bert − base − uncased* as the query encoder. We sample 10K data from remaining days for evaluation.

Table 2: The recall results in % for AOL dataset at different ks

Model	R@100	R@50	R@10
MPC (Baseline)	60.9	53.4	35.6
MPC + Semantic	67.0	61.6	47.0
MONR	**69.7**	**63.6**	**49.8**

5.1 Recall Improvement

We clearly notice a recall improvement at all ks in both datasets. The recall improves with increase in k as expected. In both the results, we observe recall at 50 using our method is higher than recall at 100 using MPC. This suggests that reducing the number of retrieved candidates can still maintain, if not significantly improve, the recall, while also saving latency during the ranking step. This will also increase budget to use more sophisticated ranking models to increase precision at top-k.

We also notice that for e-commerce dataset, MPC+Semantic recall is lower than MPC for the same k. This is because semantic candidate generator retrieves top-$k/2$ candidates only based on similarity with previous query of the user and combines with top-$k/2$ MPC candidates. It does not necessarily respect the prefix, causing decrease in recall. For AOL, dataset on the other hand, customer usually search the same query again after a few minutes. Therefore, retrieving semantically similar queries (exactly the previous query) is useful and improves recall over MPC only.

5.2 Prefix length analysis

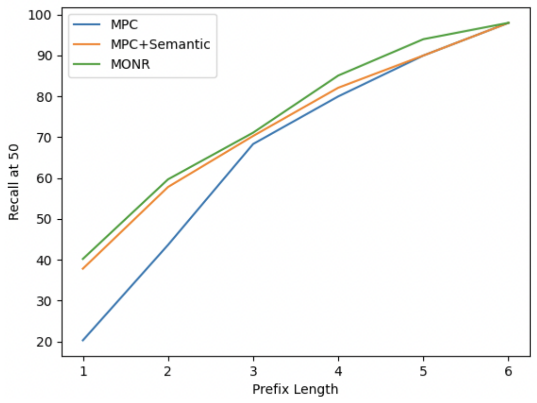

Figure 6: Recall vs Prefix length

From Figure 6, we can see the improvement in recall compared to MPC baseline is higher at shorter prefix lengths, and improvement over MPC diminishes as prefix length increases. This is because number of possible candidates for a given prefix is inversely proportional to prefix length. For shorter prefixes, as the possible number of candidates in the entire universe of queries is huge, the likelihood of top-k MPC candidates containing semantically similar queries to previous query in the session is low. As prefix length increases, MPC retrieved candidates set and MONR candidates have increasingly more common queries. This verifies our hypothesis that MONR will improve user experience at short prefixes and reduce typing efforts.

5.3 Retrieval Score as ranker

Our method retrieves top-k candidates based on highest dot product with the input vector. The dot product itself like described in equation 1 is a linear combination of three scores. We argue it can be used as a ranker. We compute mean reciprocal rank (MRR) on AOL dataset to measure how high the submitted query appears in the list if ordered by the retrieval score. We observe that MRR with our method (0.174) is higher than MRR with MPC algorithm (0.045).

5.4 Latency Discussion

Our method has higher latency compared to MPC since it includes encoding the previous query and performing FAISS search. QAC systems usually employ a neural network based ranking model after candidate generation which typically involves encoding the previous query. We argue that when comparing end to end latency, our method adds latency only due to FAISS search step. This is usually less than 1ms [7] for vectors of dimension 768 in an index containing 1 billion items. We believe the gain in recall justifies the latency increase.

6 CONCLUSION & FUTURE WORK

We presented an approach to perform multi objective neural retrieval for session aware candidate generation in QAC e-commerce setting. We used the last query searched by the user as session context. This can be further extended to include more search history as well as other user activities like products viewed and purchased to retrieve more personalized query suggestions. We used a shared BERT based encoder for both last query and next query candidates. This can be improved using ColBERT [10] architecture get more fine-grained embeddings of user activity. We can also extend the method to include different objectives like purchase conversion rate, trending score or filter queries based on a certain category like electronics, home improvement etc. In summary, our method shows that building one multi-objective vector index allows retrieving query suggestions for various objectives.

REFERENCES

[1] 1993. Okapi at TREC. 500207 (1993), 109–123.
[2] Ziv Bar-Yossef and Naama Kraus. 2011. Context-Sensitive Query Auto-Completion. In *Proceedings of the 20th International Conference on World Wide Web* (Hyderabad, India) *(WWW '11)*. Association for Computing Machinery, New York, NY, USA, 107–116. https://doi.org/10.1145/1963405.1963424
[3] Adam Block, Rahul Kidambi, Daniel N. Hill, Thorsten Joachims, and Inderjit S. Dhillon. 2022. Counterfactual Learning To Rank for Utility-Maximizing Query Autocompletion. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. https://doi.org/10.1145/3477495.3531958
[4] Fei Cai and Maarten de Rijke. 2016. A Survey of Query Auto Completion in Information Retrieval. *Foundations and Trends® in Information Retrieval* 10, 4 (2016), 273–363. https://doi.org/10.1561/1500000055
[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805
[6] Aaron Jaech and Mari Ostendorf. 2018. Personalized Language Model for Query Auto-Completion. In *Proceedings of the 56th Annual Meeting of the*

Association for Computational Linguistics (Volume 2: Short Papers). Association for Computational Linguistics, Melbourne, Australia, 700–705. https://doi.org/10.18653/v1/P18-2111

[7] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *CoRR* abs/1702.08734 (2017). arXiv:1702.08734 http://arxiv.org/abs/1702.08734

[8] Manojkumar Rangasamy Kannadasan and Grigor Aslanyan. 2019. Personalized Query Auto-Completion Through a Lightweight Representation of the User Context. *CoRR* abs/1905.01386 (2019). arXiv:1905.01386 http://arxiv.org/abs/1905.01386

[9] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. *CoRR* abs/2004.04906 (2020). arXiv:2004.04906 https://arxiv.org/abs/2004.04906

[10] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. *CoRR* abs/2004.12832 (2020). arXiv:2004.12832 https://arxiv.org/abs/2004.12832

[11] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. 2006. A Picture of Search. In *Proceedings of the 1st International Conference on Scalable Information Systems* (Hong Kong) *(InfoScale '06)*. Association for Computing Machinery, New York, NY, USA, 1–es. https://doi.org/10.1145/1146847.1146848

[12] Milad Shokouhi. 2013. Learning to Personalize Query Auto-Completion. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Dublin, Ireland) *(SIGIR '13)*. Association for Computing Machinery, New York, NY, USA, 103–112. https://doi.org/10.1145/2484028.2484076

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762 http://arxiv.org/abs/1706.03762

[14] Kai Yuan and Da Kuang. 2021. Deep Pairwise Learning To Rank For Search Autocomplete. *CoRR* abs/2108.04976 (2021). arXiv:2108.04976 https://arxiv.org/abs/2108.04976

A APPENDIX

A.1 Character Representation Vector Construction

Here we describe how the character representation vector is constructed. Each character in the text is allotted a unique index in the vector. The value is defined by the order it appears in text, exponentially decreasing with position in text. We first create a mapping function $f(c)$ which maps each character in vocabulary (26 letters, 10 numbers and 5 symbols) to a unique integer between 0 and 41.

```
Data: text, maxLength
Result: vector of size maxLength
Initialize V with zeros = [0,0,...0];
Initialize current vector position p = 0 ;
Initialize characters added d = 0 ;
Assume character to unique integer mapping function
  f(character) ;
for all characters c in the text do
    Vector position to update = p + f(c) ;
    If p + f(c) is beyond the size of V, circle back to start ;
    V[p+f(c)] = $e^{-d}$ ;
    Increase d by 1 ;
    Update current position to p+f(c) ;
end
Return V normalized by its norm$^2$
```

Algorithm 1: Pseudo Code