Session-Aware Query Auto-completion using Extreme Multi-Label Ranking

Nishant Yadav*
University of Massachusetts Amherst

Rajat Sen*
Google Research

Daniel N. Hill
Amazon

Arya Mazumdar*
University of California, San Diego

Inderjit S. Dhillon*
Amazon & University of Texas Austin

ABSTRACT

Query auto-completion (QAC) is a fundamental feature in search engines where the task is to suggest plausible completions of a prefix typed in the search bar. Previous queries in the user session can provide useful context for the user's intent and can be leveraged to suggest auto-completions that are more relevant while adhering to the user's prefix. Such session-aware QACs can be generated by recent sequence-to-sequence deep learning models; however, these generative approaches often do not meet the stringent latency requirements of responding to each user keystroke. Moreover, these generative approaches pose the risk of showing nonsensical queries. One can pre-compute a relatively small subset of relevant queries for common prefixes and rank them based on the context. However, such an approach fails when no relevant queries for the current context are present in the pre-computed set.

In this paper, we provide a solution to this problem: we take the novel approach of modeling session-aware QAC as an eXtreme Multi-Label Ranking (XMR) problem where the input is the previous query in the session and the user's current prefix, while the output space is the set of tens of millions of queries entered by users in the recent past. We adapt a popular XMR algorithm for this purpose by proposing several modifications to the key steps in the algorithm. The proposed modifications yield a 10× improvement in terms of Mean Reciprocal Rank (MRR) over the baseline XMR approach on a public search logs dataset. We are able to maintain an inference latency of less than 10 ms while still using session context. When compared against baseline models of acceptable latency, we observed a 33% improvement in MRR for short prefixes of up to 3 characters. Moreover, our model yielded a statistically significant improvement of 2.81% over a production QAC system in terms of suggestion acceptance rate, when deployed on the search bar of an online shopping store as part of an A/B test.

*Work done while at Amazon

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467087

CCS CONCEPTS

• **Information systems** → **Personalization**; • **Computing methodologies** → **Ranking**;

KEYWORDS

eXtreme multi-label ranking, auto-complete, session-aware

ACM Reference Format:

Nishant Yadav, Rajat Sen, Daniel N. Hill, Arya Mazumdar, and Inderjit S. Dhillon. 2021. Session-Aware Query Auto-completion using Extreme Multi-Label Ranking. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3447548.3467087

1 INTRODUCTION

Query auto-completion (QAC) is an important component of modern search engines. The task is to generate query suggestions that start with the prefix typed by the user in the search bar. This is of great utility to users as it saves them the effort of typing out the complete query. QAC also helps the user to formulate a query when they are not completely sure what to search for. Often a user will make a series of queries within a small time window, referred to as a *session*. In such cases, the user's previous queries provide contextual information that can be used to make their auto-complete suggestions more relevant [2, 18, 36]. Consider an example where the user has typed `"n"` into the search bar after previously searching for `"digital camera"`. Based on the previous query, it is more likely that the user is searching for `"nikon camera"` than for `"nike shoes"`. Thus the suggestion `"nikon camera"` should be ranked higher than `"nike shoes"` in the list of suggestions. In this paper, we propose a novel method for session-aware QAC which uses contextual information from previous queries made by the user to suggest better query completions within a strict latency budget.

QAC is typically performed in a two-step retrieve-and-rank process. In the first step, a limited number of candidate queries that begin with the user's prefix are retrieved. These candidates are pre-selected from prior data based on business rules. For instance, one could select the top 100 most frequent completions for a prefix based on prior data. These retrieved queries are then ranked based on features like frequency, similarity to the previous query,

user profile, etc. [5, 34, 36, 39]. Finally, the top-k ranked queries are shown as suggestions to the user. However, this approach fails when the candidate set does not contain relevant queries for the given context. This problem is especially severe for very short prefixes where the limited and fixed candidate set is unlikely to contain relevant queries for all possible contexts.

To alleviate the shortcomings of the retrieve-and-rank framework, another line of research uses generative sequence models to *generate* potential query completions, starting from the given prefix and conditioned on previous queries [10, 17, 35, 38]. Such models leverage powerful deep learning models that can generate novel queries as well as generalize to unseen prefixes. However, these generative sequence models typically have high inference latencies that prohibit their use in real-time systems where the allowed latency budget is on the order of few milliseconds to match the user's typing pace [38]. Moreover, using a generative model increases the risk of surfacing non-sensical queries in business-critical applications.

To overcome the limitations of existing techniques while still meeting the low latency requirements, we take the novel approach of modeling session-aware query auto-completion using an *end-to-end* retrieve-and-rank framework of eXtreme Multi-label Ranking (XMR) models. XMR algorithms are designed to match and rank a list of relevant items from a huge, fixed output space containing millions of labels. This approach ranks a *context-dependent* list of candidate suggestions matched from the huge output space, in contrast to retrieve-and-rank methods with a fixed context-independent list of candidates. Also, note that the output space of suggestions in XMR can be pre-filtered to reduce the risk of surfacing non-sensical queries, unlike in generative approaches.

Session-aware QAC can be framed as a multi-label ranking task where the input is the user's prefix and previous queries, and the observed next query is the ground-truth label. Several promising approaches for XMR are based on the concept of forming a probabilistic label tree [16, 20, 33, 45, 46]. We show that such tree-based methods are especially amenable to be adapted for the QAC problem. In this work, we adapt Parabel [33], a popular tree based XMR model, for this task. In particular, we use a flexible and modular open-source XMR framework, PECOS [46], which generalizes the Parabel model. PECOS proceeds in three steps: (i) *Label Indexing*. Organize the large number of labels in a hierarchical label tree. (ii) *Matching*. Given an input context, retrieve a much smaller subset of labels of interest. (iii) *Ranking*. Rank the relevant labels to return the top-k items. PECOS leverages the hierarchical structure of label indexing to perform matching and ranking with inference time logarithmic in the total number of labels. A straightforward way to apply PECOS to our task is to use the previous query to generate a ranked list of next query suggestions and filter out any suggestion that does not match the prefix. However, using PECOS out-of-the-box performs poorly. In this work, we adapt the indexing, matching, and ranking steps in PECOS so that the model retrieves and ranks queries (labels) matching the input prefix with higher probability without compromising on suggestion quality and inference latency. In summary, the key contributions of this paper are:

- To the best of our knowledge, this is the first work to use an end-to-end retrieve-and-rank XMR framework for the session-aware query auto-complete problem.
- We propose a set of label indexing methods for tree-based XMR which are amenable to the QAC problem. In particular, one of our label indexing algorithms uses a hybrid of a trie with 2-means hierarchical clustering based on a novel position-weighted TF-IDF vectorization of the label text. This leads to an improvement of 26% in MRR on the AOL search logs dataset [31], over the default indexing scheme in PECOS.
- On the AOL search logs dataset, the proposed XMR model yields up to 10× improvement over using PECOS out-of-the-box. It outperforms all baseline approaches that meet latency requirements (less than 10 ms per inference), and outperforms all baselines when the ground-truth label is present in the output space. Our gains over baseline models are sharpest for shorter prefixes where the context information is extremely important. For prefixes of 3 or fewer characters, we see a 33% improvement over the best baseline that meets latency requirements.
- Our model yielded a statistically significant improvement of 2.81% over a production system in terms of QAC suggestion acceptance rate when deployed on the search bar of an online shopping store as part of an A/B test.

2 RELATED WORK

Traditional query auto-completion systems operate in the retrieve-and-rank framework where a fixed set of queries are retrieved given the input prefix followed by ranking based on various sources of information. The simplest yet effective approach is to suggest the top-k most frequent queries matching the input prefix [39]. Other approaches further improve over this by ranking suggestions based on previous queries, user profile, time-sensitivity, or coherence with the typed prefix [5, 34, 36, 39]. Recent work also leverages deep learning models to generate additional features such as query likelihood using language models [30], personalized language models [12, 14], or previous queries [36] to rank queries using a learning to rank framework [41]. However, these models typically rely on *heuristic* methods to generate a fixed number of candidates for all popular prefixes. Our work differs from these in that we propose an end-to-end model comprising of a *learned* candidate query retrieval which is trained *jointly* with a query ranker. Moreover, our candidate set is dynamic and is fetched from an enormous output space as a function of the context.

Another line of research uses seq2seq and language models to *generate* query suggestions given a prefix, and can generalize to unseen prefixes and potentially generate new queries. Wang et al. [38] use character-level language models to generate completions for a given prefix, additionally combining it with a spelling correction module. Dehghani et al. [10] augment a seq2seq model with attention mechanism to attend to promising parts of previous queries during decoding together with copy mechanism to copy words from previous queries. Mustar et al. [28] fine-tunes pre-trained language models such as BERT [11] to generate auto-complete

suggestions. However, unlike models proposed in this work, these models typically have high inference latency which prohibits their use in realtime systems, and could potentially generate non-sensical auto-complete suggestions.

In general, XMR models fall into three categories: 1-vs-all approaches [1, 40, 43, 44], embedding based methods [3, 7, 9, 23, 24, 26], and tree-based methods [16, 20, 33, 45, 46]. A recent work [15] formulates next query suggestion *without prefix constraints* as an XMR task, and proposes Slice, an XMR algorithm based on navigable small world graphs [24]. Slice improves on limitations of traditional related search suggestion approaches in terms of coverage and suggestion density but it is non-trivial to extend Slice to adhere to prefix constraints in a time-efficient manner for query auto-completion. Tree-based data structures such as tries and ternary trees have been widely used in query auto-completion systems [4, 13, 19, 42]. Trees offer an inductive bias for handling the variable length prefix constraint in a time and space efficient manner. For this reason, we build on top of tree-based XMR methods in this work.

3 PROBLEM FORMULATION

Users often submit a sequence of queries to the search engine in a single session. These recent queries provide useful contextual information to improve the auto-complete suggestions shown to the user. The task is as follows: *given the last query q_l made by the user, and the next query's prefix p typed by the user so far, generate a ranked list of top-k next query suggestions that match the prefix given by the user and maximize $\hat{\mathbb{P}}$(next query| previous query, prefix).* In this work we use only the most recent query in the session for generating relevant auto-complete suggestions. However, the techniques introduced can be easily extended to use more than one previous query from the user as context.

In this work, we cast session-aware query auto-completion as an extreme multi-label ranking (XMR) problem and use tree-based algorithms developed for XMR which allow time-efficient inference, while handling a large output space.

Given N training points, $\{x_i, y_i\}_{i=1}^N, x_i \in \mathbb{R}^d, y_i = \{0,1\}^L$, where $L = |\mathcal{L}|$ and $\mathcal{L}$ is the set of labels (next queries in our case), an XMR model learns a function $f : \mathbb{R}^d \to [0,1]^L$. While XMR models might be able to generate a relevance score for every label given the input, only a small subset of labels are active for a data point. Therefore, XMR models typically generate a ranked list of top-k labels, $k \ll L$, given an input. We will now describe the specific XMR algorithm that we build upon.

3.1 The PECOS Framework

In this work, we adapt the flexible and modular XMR framework called PECOS [46] for the QAC problem. One instance of the PECOS framework known as XLinear generalizes a popular tree-based XMR algorithm called Parabel [33]. We use this instance of PECOS as a starting point. In what follows PECOS will mean the XLinear version of PECOS, which is conceptually equivalent to Parabel. At a high level, PECOS has three components:

- A **label indexing** model which organizes labels in hierarchical clusters by grouping similar labels together.
- A **matching** model which retrieves a small subset of label clusters given an input.
- A **ranking** model which ranks labels in the subset of label clusters retrieved by the matching model for the given input.

PECOS organizes labels in a label tree with all the labels present in leaf nodes, and learns a probabilistic model of the joint label distribution. The label tree is created by recursively partitioning labels into k_b balanced groups until each group/cluster contains less than M labels. PECOS learns a 1-vs-all classifier at each internal node of the label tree as well as a 1-vs-all classifier for each label in each leaf node. At inference time, a given input x is routed to leaf nodes using the 1-vs-all classifiers at internal nodes. It starts from the root node and uses 1-vs-all classifiers at internal nodes to decide whether the data point should proceed further along a branch. Therefore, the data point can traverse multiple paths and reach multiple leaf nodes. In order to avoid reaching all leaf nodes in the worst case, PECOS greedily selects up to B leaf nodes, when using beam search with beam width B. Beam search starts with the root node and at each level, it evaluates the probability of reaching child nodes of each node in the beam, selects up to B most probable child nodes, and proceeds to the next level. Beam search terminates with up to B most probable leaf nodes in time $O(B\hat{D}k_b \log L)$ where $\hat{D}$ is average label feature sparsity ($\hat{D} = D$ for D-dim dense features). Finally, the relevance score of each label ℓ in the retrieved leaf nodes is calculated using its 1-vs-all classifier together with the probability of path leading to its leaf node. Since there are at most M labels in each leaf node, this step takes $O(B\hat{D}M)$ time, and the overall inference time complexity is $O(B\hat{D}(k_b \log L + M))$.

PECOS out-of-the-box for QAC. A simple way to use PECOS for this task is a two step approach of ranking followed by filtering. First, we can use PECOS to predict a ranked list of labels (next queries) given the previous query q_l as input, and use the prefix p to collect suggested next queries that match the prefix. Since the number of labels is typically of the order of millions, PECOS ranks a small subset of labels for efficiency. This small subset of labels often contains very few labels (next queries) matching the prefix when using PECOS out-of-the-box, therefore often resulting in an empty list of auto-complete suggestions. One simple solution to counter this issue is to rank a much *larger* fraction of label space by increasing beam width (B) used in beam search at test time. However, increasing B causes a corresponding increase in the inference latency, potentially rendering it unsuitable for deploying in latency-critical systems.

To alleviate this issue, we make each component of PECOS *prefix-aware* so that the model retrieves and ranks next queries (labels) matching the input prefix with higher probability without compromising on inference latency and suggestion quality. We use PrefXMRTree to refer to PECOS XMR models with label indexing, matching and ranking models proposed in this work.

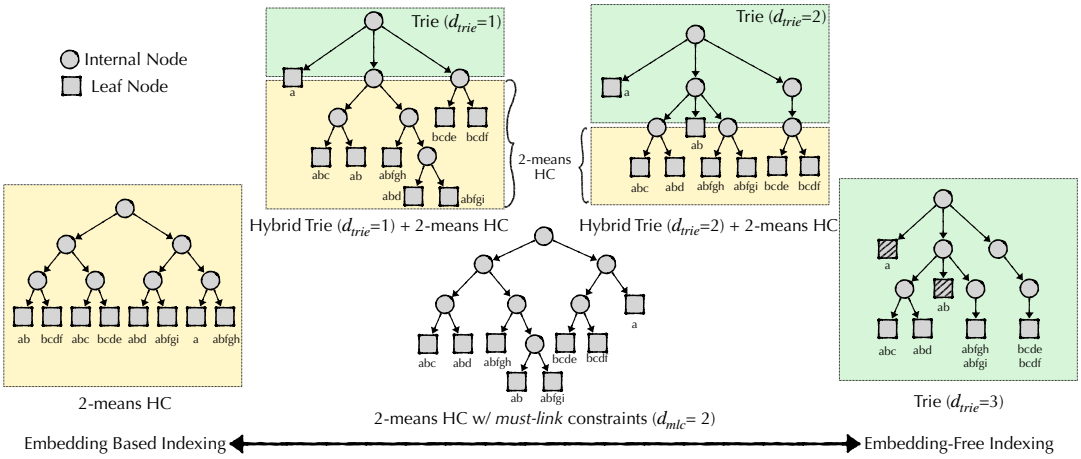

Figure 1: Label indices on a set of eight labels. We have embedding-based indexing methods such as 2-means hierarchical clustering (HC) on the left, and embedding-free indexing using a trie on the right. In the middle, we have indices which interpolate between the two extremes such as hybrid trie indices and index build using 2-means HC with *must-link* constraints.

3.2 Label Indexing in PrefXMRtree

The label indexing step in XMR tree models aims to organize labels into hierarchical clusters by grouping similar labels together. For example, Parabel indexes labels using spherical 2-means hierarchical clustering with label embeddings obtained using positive instance feature aggregation (PIFA). The matching model navigates through the index and retrieves a small subset of label clusters. Then, the ranking model ranks the labels present in those clusters. For this reason, the label index is PrefXMRtree's backbone in that the performance of the matching model, and in turn, the ranking model relies on the quality of the index to a large extent.

3.2.1 Label Indexing Algorithms. We now propose different ways of constructing tree-based indexes over the label space using approaches ranging from embedding-based methods such as hierarchical 2-means, to embedding-free approaches such as a *trie*, and hybrid approaches that interpolate between the two extremes. We also explore different label embedding strategies.

Spherical 2-means Hierarchical Clustering. This is the default indexing algorithm in PECOS. The labels are first embedded in $\mathbb{R}^p$, unit normed and then clustered with 2-means hierarchical clustering using cosine similarity subject to balance constraints. The balance constraint ensures that for a given parent cluster, the size of two child clusters output by 2-means are at most one unit apart in size. The recursive partitioning of an internal node is stopped when the node has less than M labels, and thus resulting in a cluster tree of depth $O(\log(L/M))$.

Trie. The labels (next queries) are organized in a trie data structure using the label string. The depth of a trie is proportional to length of the longest string stored in it which can be much larger than average length of its data strings. So, we limit the depth of the final trie index to a given depth d_{trie} and collapse all subtrees at depth d_{trie} to a single leaf node containing all the labels in the corresponding collapsed subtree. PECOS assumes that all the labels are stored in leaf nodes of the index tree but some labels might be stored at internal nodes of a trie. For instance, if we have two labels `"ab"` and `"abc"` and the depth of trie index is limited to 3, then `"ab"` would be stored at an internal node on path to leaf containing `"abc"`. In such cases, we create an additional dummy leaf node as a child to the internal node storing the label, `"ab"` in this example, and move the label to the dummy leaf node. In Figure 1 on the far right, we show dummy leaf nodes created for labels `"a"` and `"ab"`.

While trie index perfectly groups labels by prefixes, the resulting hierarchical index structure has a *high* branching factor, as well as a *skewed* distribution of the number of labels in each leaf node. Our initial experiments revealed that this adversely affects the training time as well as inference latency of PrefXMRtree with a trie index but it outperforms hierarchical clustering based indices in terms of output quality. The primary reason behind trie's superior statistical performance for this task is its ability to group labels with common prefixes together. On the other hand, hierarchical clustering based indices have better training time and inference latency because of their balanced structure. In order to achieve the best of both worlds, we propose the following two indexing strategies to interpolate between the two extremes.

Hierarchical Clustering with must-link *constraints.* The labels are organized using spherical 2-means hierarchical clustering (as discussed before) with additional *must-link* constraints at each level to ensure that labels with the same prefix up to length d' are present in the same cluster at depth d' of the index. The goal of these constraints is to mimic trie like behaviour of grouping labels with common prefixes, and the use of 2-means hierarchical clustering yields a balanced index with a limited branching factor of 2. Imposing *must-link* constraints for all levels of the index suffers from trie like imbalance in depth of the index. Therefore, these must-link constraints are enforced up to a fixed depth d_{mlc}. Figure 1 shows in the lower center an example of clustering with 2-means hierarchical

clustering with must-link constraints up to depth $d_{mlc} = 2$. Note that in an attempt to build a balanced tree with a fixed branching factor, it merges branches containing labels "a", and "bcde" and "bcdf". Labels with same prefix up to length 2 such as "abfgh" and "abfgi" are constrained to be in the same subtree up to depth 2 but can be present in different subtrees at depth 3 as must-link constraints are only enforced up to depth 2 in this example.

Hybrid Indexing. This approach interpolates between 2-means hierarchical clustering and trie based indexing by organizing labels in a trie of depth d_{trie}, and then further clustering labels present in each branch using 2-means hierarchical clustering. The resulting index has a relatively high branching factor and imbalanced label load distribution up to depth d_{trie} and a balanced index structure with a branching factor of 2 beyond that. Figure 1 shows a hybrid trie index in which labels are organized using a trie for a fixed depth d_{trie}= 1 and d_{trie}= 2, followed by 2-means hierarchical clustering in each subtree from depth d_{trie} onward.

3.2.2 Label Embedding. Indexing algorithms such as k-means hierarchical clustering require label embeddings in order to index them. We generate label embeddings using the following strategies:

- Positive Instance Feature Aggregation (PIFA) Embeddings: Given n training points $\{x_i, y_i\}_{i=1}^{n}$, where $x_i \in \mathbb{R}^d$, and $y_i \in \{0,1\}^L$, a label ℓ is embedded as:

  $$\phi_{\text{pifa}}(\ell) = v_\ell / \|v_\ell\|, \text{where } v_\ell = \sum_{i=1}^{N} y_{i\ell} x_i$$

  Thus, the embedding of a label is the unit-normed average embedding of training data points for which the given label is active. Different types of input embeddings (discussed in section 3.3) yield different label embeddings using PIFA.

- Label Text Embedding: Given a label ℓ, we embed the label using *character* n-gram tfidf vector of the label text. We use simple tfidf vectors [37] as well as novel position-weighted tfidf vectors for generating label embeddings from label text.

Simple tfidf vectorization of a label (next query) ignores the position of character n-grams in it. For example, "nike shoes" and "shorts nike" contain similar character n-grams but occurring at different positions. Label indexing methods using such embeddings often fail to group labels with similar prefixes together. As a result, matching and ranking models might retrieve and rank labels that do not match the prefix (but might have similar character n-grams). This hurts overall performance as most of the labels get eventually filtered out because of a prefix mismatch. Instead, we want to change the label embedding such that labels with similar prefixes are grouped together during the indexing step so that matching and ranking models are more likely to retrieve and give higher rank to labels matching the given prefix.

Position-Weighted tfidf vectors. Tfidf vectorization first counts occurences of an n-gram in the input and then multiplies it with its inverse document frequency to get the final score for each n-gram. Intuitively, we wish to assign a higher weight to n-grams which occur towards the beginning of the prefix than those which occur towards the end so that labels with similar prefixes have higher similarity than those having merely common character n-grams (possibly at different positions in the labels). We make tfidf vectorization position sensitive by modifying its counting step. Instead of each occurrence of n-gram contributing 1 unit towards its count, we modify it to contribute inversely proportional to its position. Specifically, the occurrence of an n-gram at position i contributes $1/i$ towards its count. Figure 2 shows character n-gram count vectors created during simple and position-weighted tfidf vectorization for "nike shoes", "nike shirt", and "shorts nike". Simple count vectors look similar for all three examples but position-weighted count vectors for "nike shoes" and "nike shirt" are more similar than those for "nike shoes" and "shorts nike". The final count of each n-gram thus obtained is multiplied with its inverse document frequency as in simple tfidf vectorization.

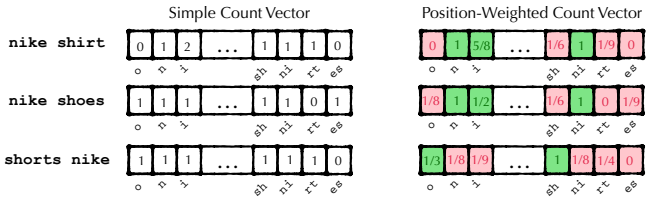

Figure 2: This example illustrates that position-weighting leads to "nike shorts" and "nike shoes" being much closer as vectors than "shorts nike".

3.3 Encoding the Input for Matching and Ranking in PrefXMRTree

The input embedding is used by the matching model to retrieve relevant label clusters and the ranking model to rank the labels in the retrieved clusters. The input consists of the previous query (q_l), and prefix (p) typed by the user. A straightforward way of featurizing the input is to only use the token-level tfidf vector of the previous query (q_l) and ignore the prefix (p).

$$x = \phi_{\text{tfidf}}(q_l)$$

However, this featurization may lead to retrieval of very few labels which match the given prefix. In order to make the matching and ranking model retrieve and rank labels starting with the given prefix with higher probability, we generate input embedding by concatenating a token based unigram tfidf embedding of q_l with a character n-gram tfidf embedding of p.

$$x = \phi_{\text{tfidf}}(q_l) \,\|\, \phi_{\text{c-tfidf}}(p)$$

For embedding the prefix, we experiment with simple tfidf vectors as well as position-weighted tfidf vectors proposed in Section 3.2.2.

4 EXPERIMENTAL RESULTS

We evaluate several PrefXMRTree models built with different combinations of label indexing methods and input embeddings on AOL search logs dataset [31] and a dataset from an online shopping store, and compare against various popular baselines. We first study the performance of an out-of-the-box PECOS model and our proposed

PrefXMRTree variants, and then compare the best performing PrefXMRTree models with generative seq2seq models, and query-frequency based auto-complete baselines.

Dataset. AOL search logs consist of sequences of queries made by the user. we lowercase all queries, replace periods with spaces, and remove non-alphanumeric characters. Further, we split a sequence of queries into sessions using a 30-min idle time between consecutive queries to define the end of a session [36, 39]. Given a session consisting of a sequence of queries $< q_1, \ldots, q_k >$, we generate $k-1$ (previous query, prefix, next query) triplets. The i^{th} triplet is (q_i, p_{i+1}, q_{i+1}), where p_{i+1} is a uniformly sampled prefix of q_{i+1}. Each such triplet serves as a data point where the input is the previous query and prefix pair and the ground-truth output is the next query. Table 1 shows statistics of train/test/dev splits for AOL search logs dataset.

We also run experiments using search logs from an online shopping store. Like AOL search logs, we split the data into train/test/dev based on timestamp, using the first few weeks' data for training, and the next few weeks' data for dev and test. The training data consist of 180MM data points with 23MM unique next queries. The test and dev set consists of 100K data points each, with 11K unique next queries of which 60% next queries were seen at training time.

Evaluation. We evaluate all proposed models using mean reciprocal rank (MRR) and reciprocal rank weighted average of the Bleu [29] score (Bleu_{RR}) of the predicted top-k auto-complete suggestions. In our experiments, we use $k = 10$.

We also measure the inference latency of each model and report 50th percentile ($p50$) and 99th percentile ($p99$) of inference latencies over the test set. All our PrefXMRTree models are trained and evaluated using a machine with Intel(R) Xeon(R) Platinum 8259CL @ 2.50GHz CPU with 16 cores. The seq2seq-GRU model is trained and evaluated on the more expensive NVIDIA V100 Tensor Core GPU with 32 GB memory.

4.1 Baselines

- seq2seq-GRU : An RNN model with GRU [8] encoder model is trained to encode the previous query q_l, and the encoded representation is fed in to an RNN decoder with GRU units to generate the next query.
- Most-Frequent Query (MFQ): This model returns top-k most frequent suggestions matching the input prefix p. This is a strong baseline corresponding to the maximum likelihood estimator for P(next query | prefix) [38]. Note that since this model completely ignores the previous query q_l, list of suggestions for each prefix can be pre-computed and stored in a database.
- MFQ + seq2seq-GRU: This model retrieves $10k$ suggestions using MFQ and uses a seq2seq-GRU model to rank suggestions based on the conditional probability of each suggestion given previous query q_l, and returns top-k suggestions.

Table 1: Statistics of AOL Search Logs Dataset

	Train	Dev	Test
Date Range (2006)	3/1 - 5/15	5/16 - 5/23	5/24 - 5/31
# Sessions	3,569,171	325,262	316,640
# Query Pairs	9,275,412	830,227	803,366
# Unique Next Queries	5,581,896	615,496	590,486

We refer the reader to Appendix A for more details on the evaluation metrics, data preprocessing, and model implementation. [1]

4.2 Comparing PrefXMRTree Models

Table 2 shows a comparison of different PrefXMRTree models on AOL search logs for different choices of input, vectorizers, and label indexing components. The baseline PrefXMRTree model (config-0 in Table 2) which uses only the previous query as input with spherical 2-means hierarchical clustering based indexing corresponds to using PECOS out-of-the-box and performs poorly. Using both the prefix and previous query as part of the input to the ranking and matching models (config-2) yields a 8.8× improvement in MRR over using PECOS out-of-the-box. For all indexing methods, label embeddings obtained using label text tfidf vectors perform better than those obtained using the PIFA strategy. The position weighted character n-gram tfidf vectorizer applied to the prefix and label yields up to 10% improvement over simple character n-gram tfidf vectorizers. Adding *must-link* constraints to 2-means hierarchical clustering for indexing yields a small improvement when also using position weighted tfidf vectorizers. This is apparently because many *must-link* constraints are already satisfied as a result of clustering with embeddings from position weighted tfidf vectorizers. The best MRR is obtained by a PrefXMRTree model that indexes labels using a trie index. However, this model takes an order of magnitude longer to train as compared to hierarchical clustering based indices because of the imbalanced structure of the trie index. The hybrid trie index structures yield a small improvement over 2-means hierarchical clustering, and perform close to a pure trie based index while having much lower training time.

4.3 Comparison with other baselines

Table 3 shows a comparison of the best performing PrefXMRTree model (config-8 from Table 2) with other baselines. Of the models that have latencies suitable for realtime systems with a latency budget of the order of few milliseconds, PrefXMRTree outperforms other baselines such as MFQ, and MFQ + seq2seq-GRU. Consistent with previous work [39], MFQ outperforms MFQ + seq2seq-GRU which re-ranks wrt likelihood conditioned on previous query using a seq2seq-GRU model (except for shorter prefixes as in Fig. 3). seq2seq-GRU outperforms PrefXMRTree in terms on MRR and Bleu_{RR} scores but has a *high* median and worst case inference latencies despite running on *GPUs*, thus rendering it unsuitable for realtime systems. Moreover, as shown in Table 4, generative models

[1] Code for reproducing experiments is available at https://github.com/amzn/pecos.

Table 2: Mean Reciprocal Rank (MRR), inference latency and training time of PrefXMRTree models for different combinations of label indexing, label embedding, and input vectorizations on the AOL search logs test set. For input embedding, we experiment with $\phi_{\textbf{tfidf}}(q_l)$ and $\phi_{\textbf{c-tfidf}}(p)$ where q_l, p denote previous query and prefix respectively, $\phi_{\textbf{tfidf}}(\cdot)$ denotes unigram tfidf vectorization, $\phi_{\textbf{c-tfidf}}(\cdot)$ denotes character n-gram tfidf vectorization, and $\|$ denotes concatenation. For vectorizing prefix or label (next query) text, we experiment with simple as well as position-weighted character n-gram tfidf vectorizer.

Config ID	Input	Prefix/Label Vectorizer	Label (ℓ) Embedding	Label Indexing	MRR	Latency (in ms) $p50, p99$	Training time (in mins)
0	$\phi_{\text{tfidf}}(q_l)$	-	$\phi_{\text{pifa}}(\ell)$	2-means HC	.0209	2.0 , 2.1	24
1	$\phi_{\text{tfidf}}(q_l)$	Simple	$\phi_{\text{c-tfidf}}(\ell_{\text{text}})$	2-means HC	.0181	2.0 , 2.2	24
2	$\phi_{\text{tfidf}}(q_l) \| \phi_{\text{c-tfidf}}(p)$	Simple	$\phi_{\text{pifa}}(\ell)$	2-means HC	.1841	7.1 , 9.6	25
3	$\phi_{\text{tfidf}}(q_l) \| \phi_{\text{c-tfidf}}(p)$	Simple	$\phi_{\text{c-tfidf}}(\ell_{\text{text}})$	2-means HC	.2088	6.8 , 9.0	26
4	$\phi_{\text{tfidf}}(q_l) \| \phi_{\text{c-tfidf}}(p)$	Pos-Wgtd	$\phi_{\text{pifa}}(\ell)$	2-means HC	.2026	6.3 , 8.1	22
5	$\phi_{\text{tfidf}}(q_l) \| \phi_{\text{c-tfidf}}(p)$	Pos-Wgtd	$\phi_{\text{c-tfidf}}(\ell_{\text{text}})$	2-means HC	.2282	6.3 , 8.2	23
6	$\phi_{\text{tfidf}}(q_l) \| \phi_{\text{c-tfidf}}(p)$	Pos-Wgtd	$\phi_{\text{c-tfidf}}(\ell_{\text{text}})$	Hybrid Trie ($d_{trie} = 1$)	.2317	5.5 , 6.5	25
7	$\phi_{\text{tfidf}}(q_l) \| \phi_{\text{c-tfidf}}(p)$	Pos-Wgtd	$\phi_{\text{c-tfidf}}(\ell_{\text{text}})$	Hybrid Trie ($d_{trie} = 2$)	.2315	5.4 , 6.5	26
8	$\phi_{\text{tfidf}}(q_l) \| \phi_{\text{c-tfidf}}(p)$	Pos-Wgtd	$\phi_{\text{c-tfidf}}(\ell_{\text{text}})$	Hybrid Trie ($d_{trie} = 3$)	.2319	5.5 , 6.6	41
9	$\phi_{\text{tfidf}}(q_l) \| \phi_{\text{c-tfidf}}(p)$	Pos-Wgtd	-	Trie ($d_{trie} = 16$)	.2220	4.9 , 5.5	140
10	$\phi_{\text{tfidf}}(q_l) \| \phi_{\text{c-tfidf}}(p)$	Simple	-	Trie ($d_{trie} = 16$)	.2310	5.1 , 6.6	137
11	$\phi_{\text{tfidf}}(q_l) \| \phi_{\text{c-tfidf}}(p)$	Pos-Wgtd	$\phi_{\text{c-tfidf}}(\ell_{\text{text}})$	2-means HC w/ MLC ($d_{mlc} = 5$)	.2291	6.0 , 7.6	24

Table 3: Mean Reciprocal Rank (MRR), Bleu_{RR} scores and inference latency (in ms) of the top performing PrefXMRTree model (config-8 from Table 2) and baseline models on entire AOL search logs test set, and on test data points for which next query is seen during training.

Model	All Test Data			Test Data w/ Seen Next Queries	
	MRR	Bleu_{RR}	Latency $p50, p99$	MRR	Bleu_{RR}
PrefXMRTree c-8	.231	.102	**5.5 , 6.5**	**.572**	**.177**
seq2seq-GRU	**.380**	**.211**	15.2, 22.1	.544	.165
MFQ	.225	.109	**0.01, 0.01**	.556	.147
MFQ + seq2seq-GRU	.173	.089	7.8, 9.1	.426	.125

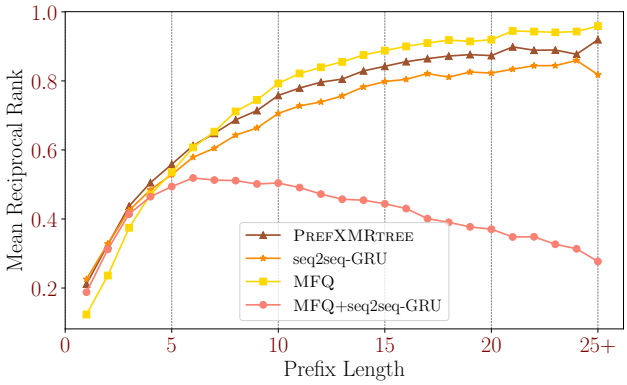

Figure 3: MRR of PrefXMRTree (c-8 in Table 2) and baselines for different prefix lengths on AOL search logs test set where next query is seen at train time.

like seq2seq-GRU can generate nonsensical auto-complete suggestions which could be undesirable in business critical applications.

The primary reason for lower scores of PrefXMRTree models as compared to generative models is the finite label space derived from the training dataset. For AOL search logs, the training data contains 5.5MM unique labels (next queries) which cover about 41% of the test data point labels. Hence, for the remaining test data points, PrefXMRTree models and MFQ cannot generate correct suggestions, while seq2seq models are sometimes able to. The performance of PrefXMRTree can be improved by augmenting the label space to improve coverage and/or using a partial prefix for generating suggestions [25, 27] but we leave that to future work and compare all the models on test data restricted to cases where the ground-truth label (next query) is present in training data, and analyse the effect of prefix length and label frequency on performance of all models.

Table 3 shows the performance of the best performing PrefXMRTree model (config-8 from Table 2) and baselines on the restricted test data. PrefXMRTree outperforms all the baselines in terms of MRR and Bleu_{RR} scores. MFQ, which corresponds to the maximum likelihood estimator for P(next query | prefix) [38], serves as a strong baseline. Note that, to its advantage, MFQ has direct access to a fixed set of labels (next queries) that exactly match the prefix whereas PrefXMRTree attempts to find and rank relevant labels from the *entire* output space containing millions of labels.

Figure 3 shows MRR for all models for different prefix lengths on the restricted test data. For all models, MRR improves with prefix length, matching the intuition that it becomes easier to predict the next query when the model has access to a larger prefix of it. Session-aware models such as PrefXMRTree, seq2seq-GRU and

Table 4: Examples of nonsensical query suggestions generated by a seq2seq-GRU model for three previous query (q_l), prefix (p) pairs with grouth-truth next query ($q^\star$) from AOL search logs test set. Prefix p of next query $q^\star$ is shown in bold.

	Examples	Auto-complete suggestions using seq2seq-GRU model
q_l	tortilla whole wheat nutrition facts	**law of m**aking nutrition facts, **law of m**eals, **law of m**cdonalds, **law of m**ushroom facts, **law of m**arijuana, **law of m**ushroom
p, $q^\star$	***law of m***otion	
q_l	ebay	**cabbage machine** recipes, **cabbage machine** diet, **cabbage machine** shop, **cabbage machine** recipe, **cabbage machine** parts
p, $q^\star$	***cabbage machine*** cutter	
q_l	prepare your own wieght loss diet	**legal s**tudies, **legal s**tudies diet, **legal s**tudies wieght loss diet, **legal s**urveys, **legal s**teroids
p, $q^\star$	***legal*** steriods	

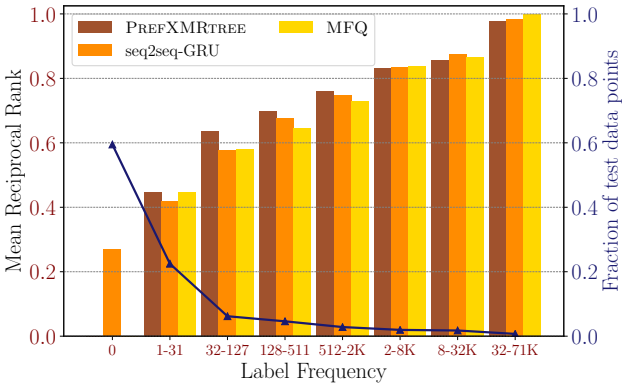

Figure 4: MRR of PrefXMRtree (c-8 in Table 2) and baselines for different label frequency ranges on AOL search logs test set where next query is seen at train time. Line plot on secondary y-axis shows the fraction of test data having labels with frequency in a given frequency range.

MFQ + seq2seq-GRU outperform MFQ for prefixes up to length 6 indicating that the context information from previous queries is most useful when the user has typed just a few characters of the next query. PrefXMRtree offers 71%, 38%, 17%, 7.3%, 4.3%, 1% improvement over MFQ for prefix lengths 1,2,3,4,5 and 6 respectively, and has the best inference latency among session-aware models.

Figure 4 shows the performance of all models bucketed by label frequency on the restricted test data and the fraction of test data with the label in a particular label frequency bucket. Generally, the performance of all models improves with label frequency. Note that a significant fraction of test data contains queries with frequency less than 2,000, and on average, PrefXMRtree outperforms MFQ for labels with frequency up to 2,000 while MFQ performs better for labels with frequency higher than 2,000.

4.4 Results on dataset from an Online Shopping Store

Table 5 shows the percentage change in MRR for different models over MFQ baseline on search logs from an Online Shopping Store. We omit MFQ + seq2seq-GRU baseline because of its poor performance on AOL search logs. seq2seq-GRU performs the best on entire test data but despite using GPUs, its inference latency and throughput are *unsuitable* for real-time systems where the latency budget is of the order of milliseconds. When the next query is present in training data, PrefXMRtree offers a significant 17% improvement over seq2seq-GRU, and 1% improvement over MFQ. A further breakdown of MRR based on prefix length shows that PrefXMRtree offers up to 143% improvement for prefix length 1, and 24% improvement over MFQ for small prefixes up to length 8.

Table 5: Percentage change in MRR for PrefXMRtree (c-5 from Table 2) and seq2seq-GRU over MFQ baseline on test set from search logs of an online shopping store.

Model	All Test Data	Test Data w/ Seen Next Queries			
		Prefix Length			
		All	1	2-8	≥ 9
seq2seq-GRU	**+37**	-12.5	+136.8	+6.0	-24.7
PrefXMRtree	+1	**+1**	**+143.2**	**+24**	-14.3

A version of PrefXMRtree was deployed online on the search bar of the Online Shopping Store as part of an A/B test, where it yielded a statistically significant improvement of 2.81% over the production system in terms of QAC suggestion acceptance rate. For reasons discussed before such as high inference latency despite running on the more expensive GPUs and the potential risk of generating nonsensical auto-complete suggestions, we could not deploy seq2seq-GRU model as part of the A/B test.

5 CONCLUSION

In this paper, we propose PrefXMRtree, a tree-based XMR model for the task of session-aware query auto-completion under stringent latency requirements. We propose and benchmark several novel label indexing and embedding schemes which make the PrefXMRtree models more suitable than an out-of-the-box XMR model for retrieving relevant suggestions conditioned on the previous query while adhering to the user's prefix. Overall, the proposed XMR models provide a better combination of statistical performance and inference latency as compared to other baselines while specifically improving suggestions for smaller prefix lengths, where the contextual information from previous queries has the most utility.

In the future, we plan to improve the performance of PrefXMRtree models by exploring label space augmentation techniques and using partial prefixes for generating auto-complete suggestions [25, 27]. We also plan to use dense embeddings for representing contextual information such as previous queries, the user-profile, or time to further improve the performance of PrefXMRtree models for the task of query auto-completion.

REFERENCES

[1] Rohit Babbar and Bernhard Schölkopf. 2017. Dismec: Distributed sparse machines for extreme multi-label classification. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*. 721–729.

[2] Ziv Bar-Yossef and Naama Kraus. 2011. Context-sensitive query auto-completion. In *Proceedings of the 20th International Conference on World Wide Web*. 107–116.

[3] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. 2015. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*. 730–738.

[4] Fei Cai and Maarten de Rijke. 2016. *A Survey of Query Auto Completion in Information Retrieval*. Now Publishers Inc.

[5] Fei Cai, Shangsong Liang, and Maarten De Rijke. 2014. Time-sensitive personalized query auto-completion. In *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management*. 1599–1608.

[6] Boxing Chen and Colin Cherry. 2014. A systematic comparison of smoothing techniques for sentence-level bleu. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*. 362–367.

[7] Yao-Nan Chen and Hsuan-Tien Lin. 2012. Feature-aware label space dimension reduction for multi-label classification. In *Advances in Neural Information Processing Systems*. 1529–1537.

[8] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Empirical Methods in Natural Language Processing*.

[9] Moustapha M Cisse, Nicolas Usunier, Thierry Artieres, and Patrick Gallinari. 2013. Robust bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems*. 1851–1859.

[10] 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 26th ACM International Conference on Information and Knowledge Management*. 1747–1756.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. (2019), 4171–4186.

[12] Nicolas Fiorini and Zhiyong Lu. 2018. Personalized neural language models for real-world query auto completion. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 208–215.

[13] Bo-June Hsu and Giuseppe Ottaviano. 2013. Space-efficient data structures for top-k completion. In *Proceedings of the 22nd International Conference on World Wide Web*. 583–594.

[14] 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*. 700–705.

[15] Himanshu Jain, Venkatesh Balasubramanian, Bhanu Chunduri, and Manik Varma. 2019. Slice: Scalable linear extreme classifiers trained on 100 million labels for related searches. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*. 528–536.

[16] Kalina Jasinska, Krzysztof Dembczynski, Róbert Busa-Fekete, Karlson Pfannschmidt, Timo Klerx, and Eyke Hullermeier. 2016. Extreme F-measure maximization using sparse probability estimates. In *International Conference on Machine Learning*. 1435–1444.

[17] Danyang Jiang, Wanyu Chen, Fei Cai, and Honghui Chen. 2018. Neural Attentive Personalization Model for Query Auto-Completion. In *IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference*. 725–730.

[18] Jyun-Yu Jiang, Yen-Yu Ke, Pao-Yu Chien, and Pu-Jen Cheng. 2014. Learning user reformulation behavior for query auto-completion. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 445–454.

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

[20] Sujay Khandagale, Han Xiao, and Rohit Babbar. 2020. Bonsai: diverse and shallow trees for extreme multi-label classification. *Machine Learning* (2020), 1–21.

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

[22] Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *Empirical Methods in Natural Language Processing*.

[23] Zijia Lin, Guiguang Ding, Mingqing Hu, and Jianmin Wang. 2014. Multi-label classification via feature-aware implicit label space encoding. In *International Conference on Machine Learning*. 325–333.

[24] Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* (2018).

[25] David Maxwell, Peter Bailey, and David Hawking. 2017. Large-scale generative query autocompletion. In *Proceedings of the 22nd Australasian Document Computing Symposium*. 1–8.

[26] Paul Mineiro and Nikos Karampatziakis. 2015. Fast label embeddings for extremely large output spaces. *arXiv preprint arXiv:1503.08873* (2015).

[27] Bhaskar Mitra and Nick Craswell. 2015. Query auto-completion for rare prefixes. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*. 1755–1758.

[28] Agnès Mustar, Sylvain Lamprier, and Benjamin Piwowarski. 2020. Using BERT and BART for Query Suggestion. In *Joint Conference of the Information Retrieval Communities in Europe*.

[29] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. 311–318.

[30] Dae Hoon Park and Rikio Chiba. 2017. A neural language model for query auto-completion. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1189–1192.

[31] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. 2006. A picture of search. In *Proceedings of the 1st International Conference on Scalable Information Systems*.

[32] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of Machine Learning Research* 12 (2011), 2825–2830.

[33] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. 2018. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 27th International Conference on World Wide Web*. 993–1002.

[34] 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*. 103–112.

[35] Jun Song, Jun Xiao, Fei Wu, Haishan Wu, Tong Zhang, Zhongfei Mark Zhang, and Wenwu Zhu. 2017. Hierarchical contextual attention recurrent neural network for map query suggestion. *IEEE Transactions on Knowledge and Data Engineering* 29, 9 (2017), 1888–1901.

[36] 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*. 553–562.

[37] Karen Sparck Jones. 1988. *A Statistical Interpretation of Term Specificity and Its Application in Retrieval*. 132–142.

[38] Po-Wei Wang, Huan Zhang, Vijai Mohan, Inderjit S Dhillon, and J Zico Kolter. 2018. Realtime query completion via deep language models. In *eCOM@ SIGIR*.

[39] Sida Wang, Weiwei Guo, Huiji Gao, and Bo Long. 2020. Efficient Neural Query Auto Completion. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. 2797–2804.

[40] Jason Weston, Ameesh Makadia, and Hector Yee. 2013. Label partitioning for sublinear ranking. In *International Conference on Machine Learning*. 181–189.

[41] Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Information Retrieval* 13, 3 (2010), 254–270.

[42] Chuan Xiao, Jianbin Qin, Wei Wang, Yoshiharu Ishikawa, Koji Tsuda, and Kunihiko Sadakane. 2013. Efficient error-tolerant query autocompletion. *Proceedings of the VLDB Endowment* 6, 6 (2013), 373–384.

[43] Ian EH Yen, Xiangru Huang, Wei Dai, Pradeep Ravikumar, Inderjit Dhillon, and Eric Xing. 2017. Ppdsparse: A parallel primal-dual sparse method for extreme classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 545–553.

[44] Ian En-Hsu Yen, Xiangru Huang, Pradeep Ravikumar, Kai Zhong, and Inderjit Dhillon. 2016. Pd-sparse: A primal and dual sparse approach to extreme multiclass and multilabel classification. In *International Conference on Machine Learning*. 3069–3077.

[45] Ronghui You, Zihan Zhang, Ziye Wang, Suyang Dai, Hiroshi Mamitsuka, and Shanfeng Zhu. 2019. Attentionxml: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification. In *Advances in Neural Information Processing Systems*. 5820–5830.

[46] Hsiang-Fu Yu, Kai Zhong, and Inderjit S Dhillon. 2020. PECOS: Prediction for Enormous and Correlated Output Spaces. *arXiv preprint arXiv:2010.05878* (2020).

A IMPLEMENTATION DETAILS

A.1 Evaluation metric details

Mean Reciprocal Rank (MRR) is the average of reciprocal ranks of the ground-truth next query in the ranked list of top-k suggestions.

$$MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{r_i}$$

where r_i is rank of ground-truth next query ($q_i^\star$) in top-k suggestions for i^{th} data point. If $q_i^\star$ is not present in top-k suggestions, then $r_i = \infty$.

$\textsc{Bleu}_{\text{RR}}$ is defined as reciprocal rank weighted average of the Bleu score [29] between the ground-truth next query and suggestions.

$$\textsc{Bleu}_{\text{RR}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\sum_{j=1}^{k}\frac{1}{j}\textsc{Bleu}(q_i^\star, \hat{q}_{i,j})}{\sum_{j=1}^{k}\frac{1}{j}}$$

where $\hat{q}_{i,j}$ is j^{th} suggestion for i^{th} data point, and $q_i^\star$ is the ground-truth next query for i^{th} data point. Bleu score between $q_i^\star$ and $\hat{q}_{i,j}$ is computed using token-level n-gram overlap. We use up to 4-gram and use smoothing method-1 from Chen and Cherry [6] to handle zero n-gram overlaps.

A.2 Implementation details for all models

The seq2seq-GRU model is trained to maximize the likelihood of ground-truth next query given the previous query. At test time, the top-k next query suggestions are generated using beam search with the constraint that the generated suggestions start with the input prefix p, and have the maximum length of 10 tokens.

We use a beam width of 16 and return top-10 of generated queries as auto-complete suggestions. We use SentencePiece tokenization [22] to create a shared vocabulary for encoder and decoder with a vocabulary size of 32,000. We use 256 dimensional input embedding, 1,000 dimensional hidden embedding, and share embeddings between encoder and decoder models. We optimize using Adam [21] with learning rate = 0.001.

For all variants of the proposed PrefXMRtree models, we encode the previous query (q_I) using word token based unigram tfidf vectorization, and we encode prefix (p) and label (next query $q^\star$) using tfidf vectorization with character unigrams, bigrams and trigrams. The vectorizers are trained using scikit-learn's TFIDF module [32]. The Position-Weighted vectorization scheme is implemented separately by modifying scikit-learn.

For training PrefXMRtree models, we use PECOS, a modular Extreme Multi-Label Ranking (XMR) framework. We use our proposed label indexing variants encoded in tree data structures for the indexing step in PECOS. We use the linear 1-vs-all classifiers at each internal node for matching and linear 1-vs-all classifiers per label for ranking (XLINEAR mode in PECOS package). The individual classifiers are trained using ℓ_2 hinge loss commonly used in linear SVM training with real positive examples and derived hard negative examples. For more details, we refer the reader to Yu et al. [46].

Code and scripts to reproduce results for PrefXMRtree models is available at https://github.com/amzn/pecos.