ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA241061

Graphite: A Graph-Based Extreme Multi-Label Short Text Classifier for Keyphrase Recommendation

Ashirbad Mishra[a], Soumik Dey[b], Jinyu Zhao[b], Marshall Wu[b], Binbin Li[b] and Kamesh Madduri[a]

[a]The Pennsylvania State University
[b]eBay Inc.

Abstract. *Keyphrase Recommendation* has been a pivotal problem in advertising and e-commerce where advertisers/sellers are recommended keyphrases (search queries) to bid on to increase their sales. It is a challenging task due to the plethora of items shown on online platforms and various possible queries that users search while showing varying interest in the displayed items. Moreover, query/keyphrase recommendations need to be made in real-time and in a resource-constrained environment. This problem can be framed as an Extreme Multi-label (XML) Short text classification by tagging the input text with keywords as labels. Traditional neural network models are either infeasible or have slower inference latency due to large label spaces. We present *Graphite*, a graph-based classifier model that provides real-time keyphrase recommendations that are on par with standard text classification models. Furthermore, *it doesn't utilize GPU resources*, which can be limited in production environments. Due to its *lightweight nature* and *smaller footprint*, it can train on very large datasets, where state-of-the-art XML models fail due to extreme resource requirements. Graphite is deterministic, transparent, and intrinsically more interpretable than neural network-based models. We present a comprehensive analysis of our model's performance across forty categories spanning eBay's English-speaking sites.

1 Introduction

In the online search space, the sponsored search mechanism [18] promotes paid entities to be shown above or beside the standard search results retrieved for a user query. The promoted entity can be shown as an advertisement to the web user or as an e-commerce listing to a buyer corresponding to their search queries. The search engine platform facilitates advertiser/seller *bidding* on search queries. The recommendation engine automatically ranks relevant search queries given a single or a group of e-commerge listings. Advertisers or sellers bid via large-scale *campaigns* to increase the visibility of their products. The solution to the recommendation task will be advantageous to both small and large-scale advertisers/sellers increasing their sales and the platform's revenues.

Query (or keyphrase) recommendations can easily be enabled from short texts representing the entity being sold/advertised, such as titles of e-commerce products, product reviews, transcripts of promotional videos, posts on social media sites, and other search-based platforms using sponsored entities [5, 24]. Here, we focus on e-commerce platforms such as Amazon, eBay, and Walmart, where the sellers are recommended keyphrases based on their inventory listing's metadata. Domain experts have found that a listing/item's title is paramount as they are quite specific, and other metadata can be missing or incoherent. For instance, a seller may have a listing titled *"New iPhone 15 Pro Max 128 GB White"* in the Electronics category. We seek to develop a recommender algorithm that will generate a relevant collection of keyphrases such as *"latest iphone"*, *"128 gb iphone 15"*, *"new iphone"*, *"apple phone 15"*, or even *"latest Samsung smartphone"*. We use *keyphrase* in this paper instead of the more common *keyword* to indicate that a phrase can have multiple words; also, the ordering of words in the phrase matters. The listing's length in words is typically very low, and the keyphrase may contain words that do not even occur in the listing.

Conventionally, logs generated by search engine responses [6, 5] to buyer queries have been the data source for recommendation systems. This helps to associate keyphrases that are not only relevant to the products, but are also actively searched by buyers. We have provided more details on the data generation process in Section 4.3.1. Aggregated data from the logs typically contains billions of data points due to the large number of *tail keyphrases* (queries searched less often). Moreover, relative to the size of the data, there are limited resources for the execution of recommendation systems. Scenarios such as the new setup of a seller's inventory require real-time or near-real-time recommendation. Thus, effective strategies should be able to suggest a small subset of relevant keyphrases from a large space of keyphrases in a constrained environment with suggestions provided in real-time.

Advertisers only want to bid on keyphrases that are actual queries and not queries that seem plausible but non-existent for targeting purposes. Since the nature of the problem is mapping items to *multiple* queries to increase the potential reach of the advertisers, this problem of keyphrase recommendation can be formulated as an Extreme Multi-Label (XML) Classification problem. Keyphrase recommendation has been explored using query-query similarity with click graphs [3], and formulated as an XML problem in [1].

Typically, the keyphrase-recommendation datasets in the advertisement domain exhibit a *power-law distribution*, with a large number of tail keyphrases. Therefore, it is difficult to cluster words that occur within listings in a category, diminishing the effectiveness of many text classifiers. On the other hand, the text classifiers that provide effective predictions have high inference cost, which is usually the issue with Large Language Models (LLMs) [21] such as BERT [14], GPT-3 [10], etc.

The datasets under consideration are provided by eBay from their proprietary search logs. The "Very large" categories in the dataset

have more than two million of keyphrases associated with more than three million training data points. Hence, the coverage of keyphrases in the training data is extremely sparse. With more products being added to the platform every second along with buyers constantly querying for new products, the datasets are expanding at an accelerated pace. To match this increase, models need to exhibit a low memory footprint and faster execution in both training and inference, while also scaling well according to the number of labels and training data points.

Our contributions and scope in this work are as follows:

- We develop a novel bipartite graph-based model that is simple and interpretable for commercial purposes.
- Our developed model is the most efficient among all the state-of-the-art (SOTA) models for keyphrase/query recommendation in the sponsored search domain with real-time inferencing. It has a low memory footprint, extremely low training time, and parallelized inferencing that scales efficiently and reliably for very large datasets with millions of labels. Also, under resource-constrained settings, it achieves the highest accuracy among the SOTA models.
- We give performance comparisons on real-world search-related datasets from the eBay e-commerce platform and public datasets. Our model handles various practical hurdles such as the *cold start* issue, optimized resource usage for real-time utility, and integration complexity with existing pipelines.
- We show some commercial impacts of the model in real-world scenarios.

2 Related Work

Bipartite graphs have long been used in numerous domains to model user search queries from the logs such as query-url graphs [6, 4] and query-ad graphs [2]. Generally, methods operating on these bipartite graphs compute similarities between queries based on the items they are associated with, which are then used to recommend the queries for new items. Simrank++ [3] improves the query similarity technique by decreasing the number of iterations needed for convergence. In addition, it improves the generated score by multiplying them by a factor depending on the number of common neighbors of those queries. However, in the worst case, such approaches would require a comparison between each pair of queries (i.e. quadratic scaling), which is infeasible when there are a large number of keyphrases. In addition, the ranking of recommended queries associated with similar items based on the relevance to the item is an issue.

The state-of-the-art models for XML problems [12, 17, 13, 25, 11] use deep neural networks (DNNs), typically with one-vs-all (OVA) classifiers. While the models [12, 13] require label features to work with, *AttentionXML* [25], *Renee* [17] and *DeepXML/Astec* [11] can work without them. Among the DNN models in the benchmarked studies [7], we find that the DeepXML/Astec model [11] is able to scale to large datasets (e.g., the public AmazonTitles-3M dataset) and has lower training time compared to competing methods. Also, the authors of [11] show that Astec achieves real-time inference latencies. Therefore, given the scope of this work, we find Astec to be the most suitable for comparison.

DeepXML/Astec [11] is a pipelined framework that processes the task of classifying texts end-to-end. Each component of the framework can be replaced to mimic different classification algorithms. Unlike another extreme classifier Slice [16], Astec generates its own embedding, which is otherwise expensive to compute. In terms of implementation and training, DeepXML consists of 3 stages. The first stage trains a *surrogate task* that generates an intermediate representation. Faster training is enabled by reducing the label space, which is done by first generating label representations from the representations of input texts (instance) that the label is associated with in the training set. Clusters are generated by associating together labels with similar representations and annotating each cluster as a meta-label. The second stage (called *extreme*) trains OVA classifiers for each label and optimizes training by *shortlisting* labels for each data point using negative sampling. The third stage called *reranker*, is similar to the *extreme* stage, which uses the pre-trained shortlists from the extreme stage.

The fastText [8, 20] software tool/model has proven to be a CPU-based, efficient solution for handling large workloads. fastText generates word vectors using the *CBOW* model and uses a simple linear neural network model with hierarchical softmax for faster training and inference. One of the reasons why fastText works well is because it incorporates *subword* information into its embeddings. The size of the model can easily be constrained to use less storage space using techniques such as quantization [19], and pruning the vocabularies of keyphrases and title words.

fastText is *one of the models deployed at eBay* for real-time keyphrase recommendation to sellers. Hence, we use it as a baseline for our comparison.

3 Graphite Model

In this section, we introduce some notation and formally define the problem we are trying to solve. Subsequently, we describe the two essential steps of our model: the *Construction* step equivalent to training, and the *Inference* step. And then discuss the implementation details.

3.1 Problem Formulation and Notations

We formally define the *Multi-label Text Classification* problem where there are multiple labels associated with each instance (input text). Each instance consists of a list of words from the textual data of the instance and a list of labels that will be modeled by a classifier. Such a model is generally constructed or learned from the training part of the dataset $T(A, B)$, which consists of two sets of lists A and B of equal size, representing instances and labels respectively. Each $a_i \in A$ is a list $a_i = \{w_1, w_2, ..., \}$, where each w_* is a word in the instance a_i. Similarly, each list $b_i \in B$ has the form $b_i = \{l_1, l_2, ...\}$, where each $l_* = \{w_1, w_2, ...\}$ is a label consisting of a list of words. Together, a_i and b_i denote the training sample for index i. Note that the ordering of elements in the label list is essential and list permutations create unique labels. Formally, a Bipartite Graph $G(V, E)$ has two disjoint subsets of vertices X and Y such that $X \bigcup Y = V$ and $X \bigcap Y = \phi$. The edges of the set E connect a vertex in X with a vertex in Y and there are no edges connecting the vertices within each subset. We define the function *Search, Deduplicate and Count* or $SDC(\cdot)$ which given a list of elements, counts the occurrences of each element in the list. It outputs a list of tuples of the form $(element, count)$ for each unique element in the list. We also define functions *Create Map* or $cmap(x, y)$ that maps x to y and *Get Mapping* or $gmap(x)$ that retrieves the mapping for x. $gmap(*)$ returns all the mapping as a list. In XML systems the instance is the item/document in question and the labels are the keyphrases. Graphite's main idea is that two items should be associated with the same keyphrase if they are similar. For inference, given a test item, our model identifies items from

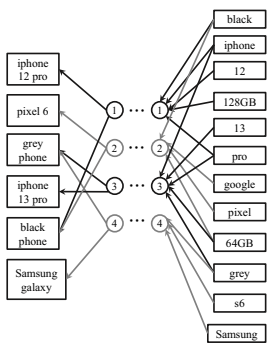
the training data that are similar to the test item. The labels associated with the similar items are then ranked in order of relevance to the test item.

3.2 Model Construction

The Graphite model constructs two bipartite graphs, $G_{WI}(V, E)$ and $G_{IL}(V, E)$ from the training set $T(A, B)$. $G_{WI}(V, E)$ maps the words within each instance/sample to the identifier (or index) of the instance/sample. Identifiers are nonnegative integer numbers of instances in the set T. The vertex set $V \in G_{WI}$ consisting of two disjoint sets, X and Y, corresponding to the unique set of words in the instance and unique instance identifiers, respectively. So, $X = \bigcup_{\forall w \in a_i, \forall a_i \in A}\{w\}, A \in T$ and $Y = \{0, 1, ..., |A|\}$, where $|\cdot|$ denotes cardinality. The edges $E \in G_{WI}$ are constructed from the tuples as $E = \{(w, i), \forall w \in a_i \in A\}$. Next, our model constructs the graph $G_{IL}(V, E)$ that maps the instance identifiers to their associated labels in T. Similarly, the disjoint subsets X and Y of the vertex set $V \in G_{IL}$ are built as follows. $X = \{0, 1, ..., |A|\}$ and $Y = \bigcup_{\forall l \in b_i, \forall b_i \in B}\{l\}, B \in T$ where each l is a label. The edge set $E \in G_{IL}$ is the unique set of tuples $E = \{(i, l), \forall l \in b_i \in B\}$. Note that the ordering of the instance identifiers does not matter.

ID	Keyphrases	Item Titles
1	iphone 12 pro, black phone	black iphone 12 pro 128GB
2	pixel 6, black phone	google pixel black 64GB
3	iphone 13 pro, grey phone	grey iphone 13 pro
4	Samsung galaxy, grey phone	Samsung s6 grey

(a) Illustrated Training Data



(b) Tripartite Graph derived from Illustrated Data

Figure 1: Illustration of Graphite's construction phase. Subfigure (a) shows a set of item's titles along with their associated keyphrase and subfigure (b) shows the graphs G_{WI} and G_{IL} that are constructed from the set in (a).

Together, both graphs G_{WI} and G_{IL} map the words in an item to the associated keyphrases akin to a tripartite graph. Modelling a tripartite graph as two bipartite graphs helps us construct and store the data efficiently. We show an illustration of the structure and contents of the graphs in figure 1. Subfigure (a) displays a set of sample training instances as the item's title along with the associated labels as keyphrases. Subfigure (b) shows the graphs G_{WI} and G_{IL} constructed from the sample training set using Graphite's construction phase. The graph G_{WI} is constructed by mapping the words in the title to the items ids (training instance ids), and the graph G_{IL} is constructed by mapping those items ids to the keyphrases. The visual shows intuitively how the keyphrases can be mapped from the words, thus, for a test instance determining relevant keyphrases is quite evident.

3.3 Inference Step

Given a test instance t with the words in the title, the classifier predicts a list of labels in order of relevance to the instance. Graphite's inference step executes two phases *Clustering* followed by *Ranking* on the test instance t which is described in the subsections below. The goal of the Clustering phase is to cluster candidate labels into groups based on their similarity to the test instance. Each candidate label is obtained from the training instances represented in G_{WI} and G_{IL}. The Ranking step re-ranks the labels in each group/cluster based on label attributes generated during the clustering phase.

Algorithm 1 Graphite's Inference

Input: Graphs G_{WI} and G_{IL} and test instance t
Output: List of lists (R) with labels and their attributes

```
1:  function CLUSTERING(G_WI, G_IL, t)
2:      I, L, R ← []                          ▷ Lists of instances, labels and results resp.
3:      for w in t do
4:          for (w, i) in E ∈ G_WI do
5:              I ← I + i
6:      I ← SDC(I)
7:      for (i, c) in I do
8:          for (i, l) in E ∈ G_IL do
9:              L ← L + l
10:             cmap(l, c)
11:     L ← SDC(L)
12:     for i ← gmax(map(∗)) to 1 do
13:         C ← []
14:         for (l, m) in L and gmap(l) == i do
15:             C ← C + (l, WMR(t, l), m)
16:         R ← R + C
17:     return R
```

3.3.1 Clustering Phase

The algorithm 1 describes the clustering phase. Given a test instance t, it generates clusters of the candidate labels as a list. Each generated candidate label is then associated with a set of attributes required for the next ranking step. The algorithm first starts by mapping the words in the test instance t to the list of instance ids (i) in G_{WI} as shown in lines 3-5 of algorithm 1. The $SDC(\cdot)$ function[1] is utilized in lines 6 and 11. The output (I) of line 6 is a list of tuples with instance IDs (i) and the occurrence count (c) of the instance. The occurrence count is termed as *similarity (score)* as it is equivalent to the number of similar words to the test instance t. Next, lines 7-10 use G_{IL} to find the labels associated with the instances in I. The similarity score (c) of each instance (i) is assigned to the label (l) associated with it using the $cmap(\cdot, \cdot)$ function[1]. Labels with two different similarity scores are mapped to the higher score. The $gmap(\cdot)$ function[1] returns the similarity score of a label and $gmap(*)$[1] returns all similarity scores. After the execution of line 11, the list L contains candidate labels with their *Multiplicity* (m) which indicates the number of unique instances from which the label was derived. The for loop in line 12 iterates from the largest similarity score to the smallest, to create a list of clusters R. Inner loop 14 ensures that each cluster only groups the labels with the same similarity score. Each element of the cluster is a tuple containing the label l and its two attributes, *Word Match Ratio* computed by the function $WMR(t, l) = \frac{|t \cap l|}{|l|}$ and *Label Multiplicity* given by m.

3.3.2 Ranking Phase

The ranking phase operates on the list of clusters R obtained at the end of the algorithm 1. The clusters in R are in non-increasing order of the similarity score, i.e, each label in the first cluster has the

[1] Defined in section 3.1

highest similarity score followed by labels with second highest similarity score and so on. This partial ordering provides us with some relevancy ranking, but the labels within each cluster are not ranked. Thus, during this phase, the labels within each cluster are ranked using their attributes. The labels are ranked in the non-increasing order of *word match ratio* and to break the ties the label with larger *label multiplicity* is ranked first. Our model follows a *label word-aware* technique that looks at the similarity between the label and the test instance. This is enabled by the *word match ratio* where labels with more common words with test instance are preferred. Note that the denominator in the function $WMR(\cdot,\cdot)$ ensures that smaller-length labels are preferred. We find that this works best for the dataset under consideration, while it can be tweaked for other datasets. *Multiplicity* of a label indicates how many similar instances were associated with the label, indicating a higher probability of relevance.

3.4 Implementation Details

The construction of both G_{WI} and G_{IL} is first done by building a list of tuples indicating the edges, then edges are sorted, de-duplicated and stored in *Compressed Sparse Row (CSR)* format. The time complexity is log-linear and the space complexity is linear in the number of edges. The number of edges is asymptotically $O(|A| \cdot \max|a_*|)$ for G_{WI} and $O(|B| \cdot \max|b_*|)$ for G_{IL}. In the implementation, the words and the labels are represented as *unsigned integers* to reduce storage costs and avoid string comparisons. In other words, comparing two words or two labels takes constant time. The training step does not involve any weight updates or hyper-parameter training, making it quite fast and efficient.

Generally, during inference, a predetermined number of labels is required to be predicted. Due to the large label spaces, our method would output a large number of predictions, many of which would be irrelevant. Hence, we limit the number of predictions by only considering the clusters in R with the highest similarity. We preemptively achieve this by modifying the function $SDC(\cdot)$ in Algorithm 1 line 6 to only include a limited number of instances in I as required by the number of predictions. Instances with higher similarity scores are only picked while ensuring that if a unique similarity score is considered then all instances with that similarity score are picked. Thus, at the end of the ranking phase, only the required number of ordered labels are returned. The list of clusters R is implemented by extending the length of each tuple to include the similarity score. Thus, the ranking phase first orders R by the similarity scores of each label and then breaks ties using both the label attributes.

The functions SDC, $cmap$ and $gmap$ are implemented using log-linear time data structures. The ranking phase sorts the list of tuples and takes log-linear time complexity. We show results in section 4.3.3 with amortized batched inferencing time. It might seem that fine-grained parallel quicksort implementation can reduce individual inference time for datasets with a very large number of training instances and unique labels. Though, they scale well for large data sizes, achieving real-time latencies with such fine-grained approaches isn't feasible. Instead, reducing the instances retrieved in I reduces the computational cost. This is because I can be quite large, especially for datasets with a large number of labels and training points. This is because certain high-frequency words can be associated with a large number of instances, thus graph G_{WI} can get quite large. This is mitigated by modifying lines 3-5 in algorithm 1 to enable intrinsic SIMD vectorization by the compiler. But the size of I input to line 6 is still large. So, during the de-duplication and counting in function SDC in line 6 we avoid log-linear time complexity by using a count array to store the occurrence count of each instance while only performing linear operations. The sizes of L and R aren't large due to the cut-off for a pre-determined number of predictions at line 6.

4 Experimentation and Results

4.1 Setup and Preprocessing

Graphite is implemented for multi-core systems and don't require a GPU. The inference part is implemented in C++ ($\geq$ g++-9.3.0) using OpenMP threading with Python wrappers using *pybind11*. We first compare our model with fastText [8, 20] as mentioned in section 2. The training was done with the best optimal hyper-parameters searched using fastText's *Automatic hyper-parameter optimization* with validation set. There was additional configuration tuning done only on the eBay datasets, such as setting the minimum frequency of words and labels, and so on. We can't discuss these details due to proprietary usage. The fastText model is based on version 0.9.2. [2]

We show comprehensive analysis with *Astec* [11] in the DeepXML framework. For each dataset, we use the configuration provided for the datasets in [7] based on similar size label space. During the training, we could only choose the *Label Clustering* configuration as other implementations weren't provided in the source code mentioned in [11]. We also test the feasibility of AttentionXML [25] on the *Very Large* categories for our eBay datasets.

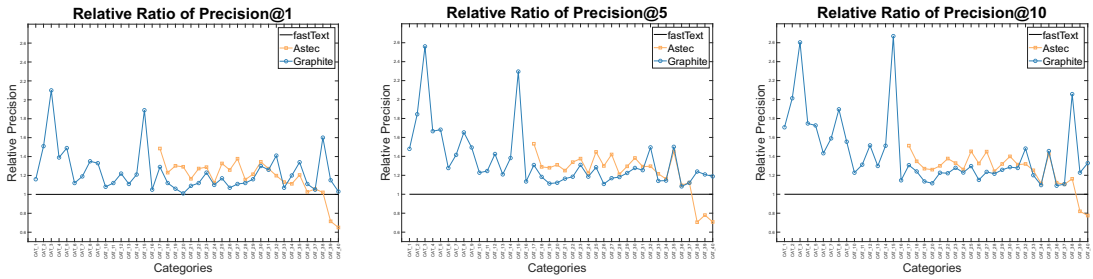
4.2 Notations and Metrics

We introduce some notations and metrics that are used for the analysis in next sections while borrowing terms from section 3.1. A test set $S(A,B)$ contains item's title words or instances (A) and keyphrases or labels (B) as lists. For each sample i, we denote the corresponding title word list as $a_i \in A$ and keyphrases as $b_i \in B$, $A, B \in S$. The a_i's are used as input and the b_i's are used as *ground truths* for the inference method described in the 3.3 to obtain the predicted labels p_i. For each sample i, we number the relevant predictions as $Relevance(i,k) = |b_i \cap p_i(k)|$, where $p_i(k)$ are the top k predicted labels. For comparison we define the metrics $Precision@k = \frac{1}{|A|}\sum_{i=1}^{|A|}\frac{Relevance(i,k)}{|p_i(k)|}$, $Recall@k = \frac{1}{|A|}\sum_{i=1}^{|A|}\frac{Relevance(i,k)}{|b_i|}$. We find that in all the datasets, a large number of items are only associated with just one keyphrase. So, the Precision@k wouldn't correctly account for the items that have different ground truth counts. Hence, we devised a metric called *Average Variable Precision (AVP)* which quantifies what fraction of ground truth on an average does the model accurately predicts. It is defined as $AVP = \frac{1}{|A|}\sum_{i=1}^{|A|}\frac{Relevance(i,|b_i|)}{|b_i|}$. For AVP calculation we place a limit of 10 ground truths, so for test data points with > 10 ground truths, only 10 predictions for each model are compared.

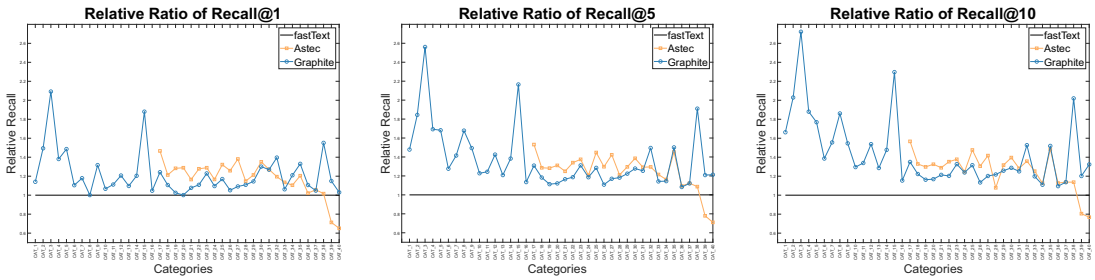
4.3 Performance on eBay Datasets

In this section, we show results on the proprietary datasets from eBay. We first describe the eBay datasets in subsection 4.3.1. The subsequent subsections compare Graphite with fastText and Astec on all the categories shown in Table 1. Due to proprietary constraints, we can't show absolute scores for the metrics described in section 4.2. Instead, we report all scores relative to fastText's performance which acts as a baseline.

[2] We used a system with 4 Intel Xeon Gold 6230 CPUs with 20@2.10GHz cores, 500 GB of RAM, and 2 Nvidia Tesla V100-32GB GPUs.



(a) Precision@1/5/10 ratios of fastText, Astec and Graphite



(b) Recall@1/5/10 ratios of fastText, Astec and Graphite

Figure 2: Comparison of Precision and Recall scores of fastText, Astec and Graphite with top 10 predictions from each model.

4.3.1 *eBay Datasets*

The datasets were generated from eBay engines' search logs. From among those items shown to a buyer, specifically the items that are clicked by the buyers are used. A buyer might input different queries/keyphrases in a session and click on some of the displayed items. Different buyers across various sessions who input the same keyphrase will be shown similar items which they eventually click on. Such occurrences of a keyphrase and the clicked item are considered as one sample keyphrase-item combination if they co-occur regularly. Similarly all such keyphrase-item associations are aggregated into a dataset if they co-occur for a sufficiently long period. It is intuitive to see how the keyphrases can be related to the items and also to the words in the titles.

Category ID	Group	# Train (in Millions)		# Labels (in Millions)	
		min	max	min	max
CAT_1-8	Very Large	3.7	25	2	7
CAT_9-16	Large	2.4	7.4	0.5	1.7
CAT_17-27	Medium	0.8	2.7	0.2	0.6
CAT_28-40	Small	0.003	0.6	0.003	0.2

Table 1: Category ID and Grouping.

The datasets consist of training/validation/test sets per top level product category in eBay. We also grouped the categories into *Very Large*, *Large*, *Medium*, and *Small* based on the number of training points and a number of labels in each category. The anonymized categories (40) which are numbered in the non-increasing order of their training size and their groups are mentioned in Table 1. The table also shows the range of the number of training data points and labels for each group. The training set contains the bulk of the historical data, while validation and test sets for tuning and testing are limited to a few thousand data points.

4.3.2 *Prediction Performance*

For each category, the top 10 predictions are used for comparison on the test set. fastText's baseline is set to 1 for all categories while Graphite's and Astec's performance is shown relative to it. We show the comparison of Relative Ratios of Precision@1/5/10 for the models in figure 2a for all categories. The categories in the x-axis follow the order of *Very Large* to *Small* group as in table 1. Graphite shows better performance than fastText for all the categories across all scores, especially with an average improvement of 122% in Precision@1 with up to 210% improvement for CAT_3. For all precision scores, the gap in the precision score is generally higher for categories with larger label spaces, while precision score gap is typically lower for categories with smaller label spaces. The Recall scores in figure 2b also show that Graphite has better performance w.r.t fastText, especially with Recall@10 scores gaining an average improvement of 140% with highest being 270%. In Precision/Recall@k with $k > 1$, the gap between Graphite and fastText is higher than the trend of $k = 1$. This is due to fastText's label space limitation which excludes tail keyphrases that Graphite is easily able to capture. The *Average Variable Precision* (AVP) scores in figure 3 show average performance akin to a precision score. Graphite performs better than fastText with an average increase of 0.085 with up to 0.207 with respect to fastText. The chart shows that Graphite on average is better able to recommend at least one of the ground truths for the items.

On the other hand, Astec fails to execute on categories CAT_1-16 in figures 2 and 3. For instance, it tries to allocate more than 1 TB of RAM for CAT_11 which was larger than the system's memory resources. The requirements are much larger for other *Large/Very Large* categories. Astec's clustering process doesn't scale well for a large number of training points. Similarly, DNNs such as AttentionXML also fail on these large datasets while allocating large GPU memory. More details regarding their executions are discussed in section 4.3.4.

Astec's performance is higher than fastText's for medium and small size categories with average P@1 and R@10 improvement of 115% and 124% respectively. However, it underperforms on very small size categories, especially on CAT_39 and CAT_40, across all precision, recall and AVP scores due to relatively smaller number of training points than the number of labels. On the categories that Astec executes on, Graphite is comparable to Astec, with an aver-

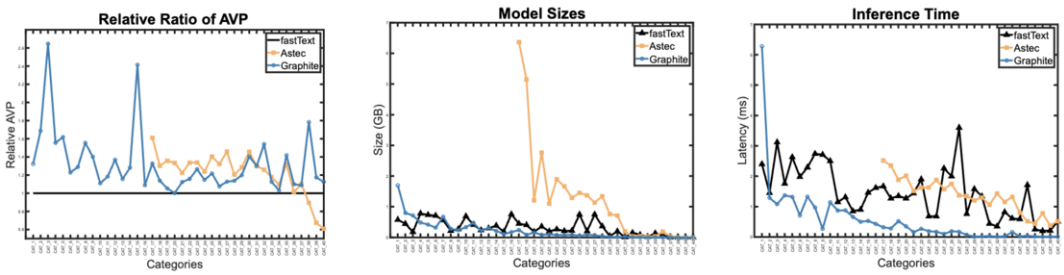


Figure 3: Comparison of AVP, Model size and Inference time ratios of fastText, Astec and Graphite

age gap of ± 0.011 between the relative performances of both the models. In contrast to Astec, Graphite generally shows consistently better performance across all data sizes, except for a few *Large* and *Medium* categories.

4.3.3 *Execution performance*

In this section, we show comparisons based on model size, inference time, and training time. If we look at the top 3 categories (CAT_1-3) with the largest label space, Graphite's trained model occupies $2-3\times$ fastText's storage on disk as shown in the middle chart of figure 3. For all the other categories Graphite's model size is comparable or occupy lower space on disk than fastText. Infact, if we sum up the model sizes of all categories, Graphite occupies 30% less space than fastText. As Graphite's storage is a linear function of both the number of training data points and unique labels, the models of larger categories occupy more space than the smaller ones. This can be tuned by eliminating rarely occurring keyphrases along with those training items that don't have any keyphrases left after the removal. This might not significantly reduce the performance when retrieving the tail keyphrases. On the other hand, as expected Astec's models are extremely large, occupying *1.2 GB* on average per category.

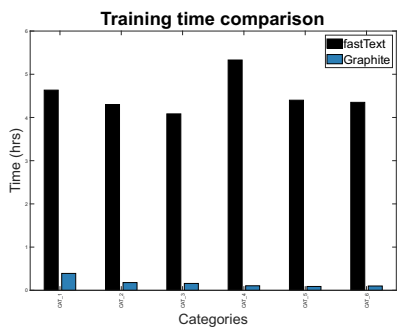


Figure 4: Training times of fastText and Graphite for the top 6 categories.

We compute the inference time per test data point by amortizing the time taken for inferencing the entire test dataset. In Graphite, we employ a coarse-grained multi-threading technique that executes individual inferences on separate threads whereas fastText is single-threaded. We launch a variable number of OMP threads (max 26) and set the number of predictions to 10. The right-most chart of figure 3 shows the per-sample inference times for fastText, Astec and Graphite on all the categories. For all except the largest category, the inference times of both models are comparable, with a geometric mean speedup of $7\times$ of Graphite over fastText. Graphite is much faster than Astec with an average speedup of $20\times$. Astec's inference was done on GPU due to absence of any CPU based code. Graphite's inference time on Small datasets group is extremely small with upto $90\times$ speed up over both fastText and Astec. Since, CAT_1 is the largest category by label space and number of items, Graphite results in a longer inference time due to the reasons discussed in section 3.4. We mitigate this practically by using a relatively smaller training set for CAT_1 corresponding to a shorter historical duration. Although this results in a smaller number of CAT_1 items, it still has sufficient number of items resulting in minimal impact on the recommendation.

Figure 4 shows the training times required to generate a model for both fastText and Graphite. We only compare using categories from the *Very Large* group from Table 1, due to their much larger training times. The *autotune* duration of fastText was set to a certain number of hours to choose the best hyperparameter for optimal model size and precision/recall scores. In some cases, getting an optimal score required autotuning for $\sim$ 10 hours. The average training time of categories on which Astec executes was 1.8 hours with up to 7 hours for some categories. This excludes the time to compute the surrogate clustering stage which runs into hours. The categories that Astec fails to execute on, would require much larger epochs resulting in training time running into days. In contrast, Graphite takes only a few minutes to generate a model and thus, enables frequent model refresh and efficient model management.

4.3.4 *Shortcomings of DNN models on Huge Datasets*

To discuss Astec's failure to execute on *Very Large* Categories, we first dive into more detail on its *surrogate* task which it's pivotal stage. Astec's surrogate stage performs two steps in conjunction, Label selection and Intermediate representation training. As described in section 2, the selection is done using clustering which requires generating a representation per label. This representation is generated from those training data points that each label is associated with. To perform the clustering, Astec tries to allocate memory in RAM as a function of the number of training points and labels. Thus, it fails for all the categories in *Very Large* and *Large* groups due to the infeasibility of the allocation for their sizes as mentioned in table 1.

AttentionXML also doesn't run for the *Very Large* categories. It encounters limitations due to its substantial CUDA memory requirements for conducting the forward pass and computing gradients during its multi-label attention phase. Even for category CAT_8, AttentionXML requires more GPU memory than two Tesla V100 (>64GB) to train the model. This requirement would be even larger for categories CAT_1-7, and theoretically for CAT_1, the requirement could exceed the specifications of commercial GPUs even when the hidden layer's size is restricted to 64.

4.3.5 *AI Evaluations and Case Study*

The keywords predicted were evaluated for a subset of randomly sampled 100 items using GPT-4 [23] evaluation as a proxy for human evaluation.[3] The responses were *yes/no* answers to relevance be-

[3] More information is available in Appendix at https://arxiv.org/abs/2407.20462.

tween the title and keywords. Table 2 shows the percentage of item-keyword pairs in comparison to fastText that were evaluated as yes for one *representative category* from each group in table 1 including the ground truth keywords for the subset of items. [4] *Graphite shows superior alignment with GPT-4 evaluation.*

Group	Category	fastText	Astec	Graphite	Ground Truth
Very Large	CAT_4	30.6	-	**61.2**	92.9
Large	CAT_11	44.6	-	**72.8**	90.9
Medium	CAT_18	61.2	76.6	**78.0**	91.9
Small	CAT_28	50.4	72.8	**75.5**	91.1

Table 2: Percentage of relevant keywords from AI evaluations of 10 predictions for 100 items from fastText, Astec and Graphite.

A case study on the item titled *"treated wooden deckboards"* demonstrates the effectiveness of each model's top recommendation. Graphite predicts *deckboards* which is the same as the ground truth and found relevant by GPT4. In contrast, fastText and Astec predict *moraea* and *wooden stakes* respectively, both being annotated irrelevant. For another item *"bird bath stone water basin garden decoration food bowl frost resistant"*, Graphite predicts *stone bird bath* while both fastText and Astec predict the irrelevant keyphrase *floor bird bath*. In both examples, Graphite's predictions have similar words to the item's title, thus our model is better able to find the nuances between different labels recommended for an input. Whereas, fastText and Astec embeddings yield irrelevant results due to imperfect learning.

4.3.6 Impact

Graphite was deployed as part of eBay's seller side keyphrase-recommendation service. This was possible due to its scalability and lightweight nature. A differential pre-post analysis was performed —which showed that Graphite was able to increase the eBay platform's coverage by recommending 6% more unique keyphrases and 17% more unique item-keyphrase pairs (keyphrase coverage). The *acceptance rate* — the fraction of keyphrases out of the recommended keyphrases accepted by the sellers to place bids on — for Graphite keyphrases was 3% higher than fastText[5]. The lift in acceptance rate reinforces the fact that *Graphite keyphrases are better in terms of seller (human) judgement than fastText*, as sellers show a greater proclivity to bid on these keywords than fastText. We cannot disclose any more information due to proprietary and business constraints.

4.4 Performance on Public Datasets

Apart from the analysis of the eBay datasets, we also wanted to compare the performances of the models on other accessible datasets for the sake of reproducibility and absolute comparison. The applicable datasets should be search-based containing queries/keyphrases that bear similarities to the input text. The label text is essential for the comparison as our model uses word matches in the label. The standard datasets in the Extreme Multi-Label Short Text classification space [7] are derived from real-world applications, ranging from item-to-item recommendation (*AmazonTitles-670K*, *AmazonTitles-3M*, etc.), to text-category tagging (*AmazonCat-13k*, *Wikipedia-500K*, etc.). However, for the purpose of keyphrase recommendation, these datasets were not suited for our analysis.

For our analysis, we hinged on publicly available keyphrase recommendation datasets available in [9]. Out of these datasets, we picked *KPTimes* [15] and *KP20k* [22] which are the largest datasets to compare for scalability. KPTimes has 102,357 unique labels associated with 259,923 training data points which are small enough for large models to handle. While KP20k has 680,117 uniques labels for 514,154 data points.

Table 3 shows the absolute performances of fastText, Astec and Graphite on KPTimes and KP20k. The Precision@5 and AVP scores of Astec for KPTimes were better than all the models while Astec couldn't provide any meaningful predictions for KP20k. For other scores, all model's performances were the same and Graphite's performance stood out in execution metrics. [6]

Metrics	KPTimes			KP20k		
	fastText	Astec	Graphite	fastText	Astec	Graphite
P@1	0.39	0.41	**0.41**	0.10	-	**0.28**
P@5	0.21	**0.25**	0.21	0.05	-	**0.12**
P@10	0.14	0.14	**0.14**	0.04	-	**0.08**
R@1	0.08	0.08	**0.08**	0.02	-	**0.05**
R@5	0.21	0.21	**0.21**	0.05	-	**0.12**
R@10	0.27	0.27	**0.27**	0.07		**0.16**
AVP	0.22	**0.25**	0.22	0.05	-	**0.13**
Inference Time (ms)	6.2	0.32	**0.03**	67.7	0.36	**0.05**
Training Time	4h	0.5h	**7.5s**	4h	1.8h	**21.2s**
Model Size (MB)	47	595	**16**	324	299	**50**

Table 3: Performance of fastText, Astec and Graphite on KPTimes and KP20k dataset.

5 Conclusions and Future Work

We present a *simple*, *transparent*, and *lightweight* graph model that can classify an extremely large number of labels in real-time. We show comparisons with baseline models like fastText and with a state-of-the-art model like Astec using real-world datasets provided by eBay. We find that fastText performs reasonably well on the extreme classification task due to its linear neural network architecture. Graphite's performance is better than that of fastText based on the Precision, Recall, and AVP scores for the 40 eBay categories that we analyzed. Both our model and fastText can handle larger datasets. Although Astec's performance is comparable to Graphite, it fails to execute on *Very Large* and *Large* categories due to its usage of *centroid* method in the surrogate task stage. Graphite has lower training time than fastText with its inference time comparable to fastText. In the future, we aim to apply Graphite to a variety of other classification tasks especially where the label text shares words with the input texts. We find that the clustering phase of our model is crucial to its performance, which can be further improved with better and light-weighted clustering algorithms. The execution and storage cost of the large category in the eBay dataset can be further mitigated by developing distillation techniques that reduce the number of training points without compromising any significant information. Graphite's bipartite graph can also include weights indicating relational propensity among the words, instances, and labels.

[4] All the GPT-4 numbers for the ground-truth are greater than 90% showing high-degree of alignment with positive buyer judgement.

[5] All these numbers were calculated over a period of one month for over millions of keyphrases and thousands of sellers.

[6] fastText's per inference time on the datasets in table 3 is relatively higher as the model's autotuning was carried out *only* using its *autotune* parameter. The tuning of parameters such as minCount and minCountLabel reduces the sizes of the token and label spaces, thus reducing inference time.

References

[1] R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the 22nd International Conference on World Wide Web (WWW '13)*, pages 13–24, 2013.

[2] T. Anastasakos, D. Hillard, S. Kshetramade, and H. Raghavan. A collaborative filtering approach to ad recommendation using the query-ad click graph. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09)*, page 1927–1930, 2009.

[3] I. Antonellis, H. Garcia-Molina, and C.-C. Chang. Simrank++ query rewriting through link analysis of the clickgraph (poster). In *Proceedings of the 17th International Conference on World Wide Web (WWW '08)*, pages 1177–1178, 2008.

[4] R. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07)*, page 76–85, 2007.

[5] K. Bartz, V. Murthi, and S. Sebastian. Logistic regression and collaborative filtering for sponsored search term recommendation. In *Second Workshop on Sponsored Search Auctions*, 2006.

[6] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*, page 407–416, 2000.

[7] K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016. URL http://manikvarma.org/downloads/XC/XMLRepository.html.

[8] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, pages 135–146, 2017.

[9] F. Boudin. https://github.com/boudinfl/ake-datasets/.

[10] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS '20))*, pages 1877–1901, 2020.

[11] K. Dahiya, D. Saini, A. Mittal, A. Shaw, K. Dave, A. Soni, H. Jain, S. Agarwal, and M. Varma. Deepxml: A deep extreme multi-label learning framework applied to short text documents. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining (WSDM '21)*, page 31–39, 2021.

[12] K. Dahiya, N. Gupta, D. Saini, A. Soni, Y. Wang, K. Dave, J. Jiao, G. K, P. Dey, A. Singh, D. Hada, V. Jain, B. Paliwal, A. Mittal, S. Mehta, R. Ramjee, S. Agarwal, P. Kar, and M. Varma. Ngame: Negative mining-aware mini-batching for extreme classification. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining (WSDM '23)*, page 258–266, 2023.

[13] K. Dahiya, S. Yadav, S. Sondhi, D. Saini, S. Mehta, J. Jiao, S. Agarwal, P. Kar, and M. Varma. Deep encoders with auxiliary parameters for extreme classification. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23)*, page 358–367, 2023.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. 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, 2019.

[15] Y. Gallina, F. Boudin, and B. Daille. KPtimes: A large-scale dataset for keyphrase generation on news documents. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 130–135, 2019.

[16] H. Jain, V. Balasubramanian, B. Chunduri, and M. Varma. Slice: Scalable linear extreme classifiers trained on 100 million labels for related searches. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining (WSDM '19)*, page 528–536, 2019.

[17] V. Jain, J. Prakash, D. Saini, J. Jiao, R. Ramjee, and M. Varma. Renee: End-to-end training of extreme classification models. *Proceedings of Machine Learning and Systems (MLSys '23')*, 2023.

[18] B. J. Jansen and T. Mullen. Sponsored search: an overview of the concept, history, and technology. *International Journal of Electronic Business*, pages 114–131, 2008.

[19] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[20] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, 2017.

[21] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*, 2023.

[22] R. Meng, S. Zhao, S. Han, D. He, P. Brusilovsky, and Y. Chi. Deep keyphrase generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 582–592, 2017.

[23] OpenAI. ChatGPT-4 (June 13 version). Large language model, 2023. URL https://chat.openai.com/chat.

[24] W.-t. Yih, J. Goodman, and V. R. Carvalho. Finding advertising keywords on web pages. In *Proceedings of the 15th International Conference on World Wide Web (WWW '06)*, pages 213–222, 2006.

[25] R. You, Z. Zhang, Z. Wang, S. Dai, H. Mamitsuka, and S. Zhu. Attentionxml: label tree-based attention-aware deep model for high-performance extreme multi-label text classification. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NIPS '19')*, pages 5820–5830, 2019.