NNCR: Revising Classifications using Embedding Based K-Nearest-Neighbor Search

Yu-Hsuan Kuo
Amazon
CA, USA
yuhsuank@amazon.com

Saaransh Gulati
Amazon
CA, USA
saaransg@amazon.com

Xiaoyu Chu
Amazon
CA, USA
xiaoychu@amazon.com

Pulkit Garg
Amazon
CA, USA
pulkitg@amazon.com

***Index Terms*—Nearest Neighbor Search, Multi-label Classifications, End-to-end Classification Revising Systems.**

The global e-commerce store needs to ensure compliance with various regulations at local, national, and international levels. One business use case is to identify face masks to avoid price gouging during times of high demand. In order to keep billions of items safe and legally compliant, it is important to ensure accurate classifications. Classification revisers aim to enhance classification accuracy by detecting and revising incorrect classifications. In this paper, we introduce this problem, along with appropriate online evaluation metrics for large-scale application scenarios. Our proposed method first learns neural network embedding from item textual features to define similar neighbors, and then simply uses these known classification results from k nearest neighbors to estimate an item's class assignment. Our experiments demonstrate that it outperforms the state-of-the-art baseline approaches and its robustness to large scales. The proposed method can uniquely revise a significant number of classifications correctly, complementary to a multi-label classification system. The study indicates that our simple yet effective approach empowered by GPU computation is a viable solution of a commercial multi-label-classification revision problem.

I. Introduction

It is essential to have accurate classification of all products for global e-commerce businesses. The applications of such classification include compliance with regulatory requirements, legality, shipping transportability, buyer age appropriateness, etc. For example, illegal and/or unsafe products should be identified; otherwise it is in violation of government regulations. Failure to detect inaccurate or missing class assignments to items has an adverse effect on business reputation and customer trust.

In a billion scale collection of categorized items, it is inevitable to have items with missing or incorrect class labels, regardless of the underlying approach for categorization. Even in the case of a model with high performance, a small percentage of inaccuracy of the model can still pose a risk. Manually reviewing all uncertain, low-confidence classifications would be impossible at this scale. Thus, an effective and scalable classification reviser to enhance and expand class labels is highly sought after.

Each product can be characterized by a set of true/false predicates over classes. We use the term ***label*** to refer to either a class being true or a class being false for a particular product. For example, a product that is a nontaxable food item should have two labels `food=True` and `is_taxable=False`. In a real-world scenario, it is possible to have over a hundred thousand classes in a classification system. To revise classifications, we formulate it as an extreme multi-label problem [1] where each product belongs to more than one class-of-interest. The classification reviser is trained to predict if a product is a member (i.e. True) or not a member (i.e. False) of a class-of-interest. If the prediction is of high confidence and different than its current assignment, we consider this product associated with this class-of-interest as a candidate for enhancement.

The standard approach is to build a multi-label classifier or multiple binary classifiers for each label. These efficient supervised models [2], [3] are still slow and computationally expensive [4] due to the extreme large label space, so algorithms aim to solve eXtreme Multilabel Ranking (XMR) problem [1], [5]–[10] have been proposed. However, XMR based methods still cannot do well for all the classes in our problem. First, our data is noisy with issues such as labeling errors and class rationale is complicated. For example, it is very challenging to classify the products between hoodie with face cover and face mask; or, learn the subtle difference between a pencil class and a pen class where they serve the same purpose as writing instruments. Second, the label distribution is long tailed and very sparse. In large-scale data, clean training data is often unavailable. Noisy labels and feature corruptions can bias a model when it is learned from noisy data [11]. Thus, the performance of multi-label

979-8-3503-2445-7/23/$31.00 ©2023 IEEE

supervised methods that are trained on the noisy, skewed, and class-imbalanced data suffers for certain classes. Instead, a scaled nearest neighbor based approach could complement these methods as it enjoys higher precision without knowing the data distribution and can generalize better from a smaller clean training set [12], [13].

However, finding similar neighbors in the large scale of hundreds of millions of items is far from trivial. One common challenge is label sparsity, where a majority of items' class assignments are unknown. Another challenge is that often the number of classifications of items that have not yet been manually verified is enormous and that the data is dynamic, i.e., the labeled data, newly added items, and the number of classes are constantly increasing. To overcome these challenges, we need a fast and robust solution to look for top k neighbors to estimate the classification results for queries and revise the classifications timely. In the spirit of "exact search is still better than the approximation" in pursuit of high recall and precision, we study whether simple yet effective methods can be empowered by the benefit of the recent success of GPU optimizations.

We propose `NNCR`, i.e., a Nearest Neighbor approach for Classification Revision which employs an efficient similarity search [14] for top neighbors using pre-trained text-based feature embeddings. By doing so, it gives an advantage in handling rare classes as our prediction is estimated by propagating the classes from similar data with known labels. More importantly, the neighbor-based approach also provides us a better interpretation of why the revision is proposed.

While many existing multi-label classifiers strive to outdo each other in different data characteristics, empirically we find that none of them outperforms the others on all cases. To achieve higher recall in identifying as many potential revision candidates as possible, different types of method can be considered in a multi-label classification system with an ensemble model [15] to consolidate the results from base learners.[1] However, in this work, we focus on the proposed neighbor-based approach in solving the extreme multi-label problem, which can serve as one of the base models.

To summarize, our contributions are as follows:

- We introduce and formulate the multi-label classification revision problem, along with appropriate online evaluation metrics for large-scale application scenarios such as e-commerce product directories.
- We propose a scalable neighbor-based end-to-end method to boost classification accuracy. To the best of our knowledge, we are the first to apply the state-of-the-art FAISS model [14] to revise extreme multi-label classifications at an industrial scale.
- We study the performance of both the k nearest neighbor (KNN) and approximate nearest neighbor (ANN) algorithms in similarity search using product embeddings.
- An evaluation on real-world data validates our approach. We conduct rigorous experiments with offline and online metrics. Our method uniquely revises a significant number of classifications correctly, as compared to an ensemble method [15].

This paper is organized as follows. We first review related work in Section II, and formally define the problem in Section III. We propose the algorithms for the classification revision in Section IV, and present experiments in Section V. Finally, we discuss the future work and conclude in Section VI.

II. Related Work

A fundamental task in the classification revision problem is detecting candidates for revision, which are data points that can either be classified more accurately and/or be tagged with additional labels. The nearest problem that involves an analogous task is anomaly detection. Therefore, in this section we review the literature for anomaly detection.

Anomaly/Outlier detection methods aim to identify anomalies with patterns that deviate from a well defined concept of normal behavior [16].

Classical methods include local outlier factor (LOF) approach [17], distance-based method [18]–[20] and Isolation Forest [21]. KNN based anomaly detection methods [22], [23] define the presumed normal data from how its neighbors are classified, and is popular in applications such as detecting anomalies in wireless sensor networks [24], system logs [25] and intrusive program behavior [26]. To this day, how to find a universal neighborhood definition to model normal behavior is still under discussion and varies largely on the type of data and problem. Also, none of these approaches can be applied to large-scale applications with over a hundred thousand classes in the systems, and explicitly consider the noise in the training data.

The work most closely related to our problem is to use deep hybrid models for identifying anomalies. The idea is taking deep neural networks as feature extractors and the features learned within the hidden representations are the input to a traditional anomaly detection algorithm [27]. Ergen et al. [28] consider the joint training of feature extractor along with one class SVM objective to maximize the detection performance. To overcome the lack of trainable objectives customized for anomaly detection, one-class neural networks are introduced. This line of work combines deep networks to extract a rich representation of data along with a one-class objective such as a hyperplane [29] or hypersphere [30] to separate normal data points from outliers. Although these methods are powerful for the problems they are targeting, they were not designed to be a general solution for our extreme multi-label classification revision problem as they still require us to build one anomaly detector per label. We argue that, compared to the literature, our customized solution is particularly tailored to our specific multi-label classification patterns, striving for both accuracy and performance under GPU environments.

[1] Alternatively, one can learn a meta model to select among different methods for each class.

III. Problem Definition

Consider a database consisting of these 3 tables: Human_Labels(entity_id, class_id, decision), Embeddings(entity_id, vector), and Inferred_Labels(entity_id, class_id, decision), where $entity\_id$ is a unique identifier for entities and $decision$ is a boolean value. The $decision$ in Human_Labels stores ground truth label generated by human annotators, while $decision$ in Inferred_Labels stores the current classification result given by models. Every entity e_i may belong to more than one class and its current inferred class information is stored in the $Inferred\_Labels$ table. The human labeled records are stored in the $Human\_Labels$ table where a record $(e_j, c_i, True/False)$ indicates that entity e_j is determined to be in class c_i (if $True$) or not (if $False$). Note that for each entity, most of its classes and the associated decisions are not available (or unknown). In a real large-scale system, it is quite common that, for each class, we only have a limited number of labeled positive and negative examples due to the cost and human bandwidth[2]. Thus, the labeled data is very sparse and skewed. For example, Human_Labels(e_1, c_1, *True*) and Human_Labels(e_1, c_2, *False*) represent that entity e_1 belongs to class c_1, but should not belong to class c_2 and its actual membership of other classes is unknown.

The feature representation of each entity e_i is encoded by its embedding vector v_i. We aim to identify the misclassified entities in the $Inferred\_Labels$ table.

Problem 1: Suppose we have a database of entities $D = \{e_i\}_{i=1}^{n}$ from Human_Labels table, where n is large, in the order of at least hundreds of millions. Given a query $q = (e_q, c_i^q, flag)$, where $flag = True/False$, our goal is to predict the decision l_i^q w.r.t. class c_i^q and $flag$ (True/False implies e_q is/is not in class c_i^q, respectively) with a confidence score s by neighboring entities in D.

An intuitive solution is to find similar entities to the query entity e_q and use all the decision set $\{l_i^s\}_{s=1}^{k}$ in $Human\_Labels$ table from its k neighboring entities $\{e_s\}_{s=1}^{k}$ to predict if e_q is or is not a member of a class c_i. The problem can be decomposed into the following sub-problems: (1) how to define similar neighbors; (2) how to aggregate the decision set from neighboring entities (i.e. resolve conflicting decisions); (3) assign a score s which indicates its estimated confidence and thus can be used in generating a revision signal.

Here, we name similar entities of query entity e_q as neighboring entities or simply neighbors. We consider an entity as a similar entity to e_q if the embedding distance between them is below a given threshold τ.

Thus, the set of similar neighbors of e_q is defined as:

$$\mathcal{N}(e_q) = \{e_p \in D | dist(v_p, v_q) \leq min(\tau, \kappa)\}, \qquad (1)$$

where $dist()$ is the distance function that takes query embedding vector v_q and neighboring entity's embedding vector v_p, and $\kappa = dist(v_k, v_q)$, the distance of the k^{th} neighbor e_k to e_q. The selection of distance function depends on the entity embedding learnt from the pre-trained task. Note that due to the scale of our problem, we do not retrieve all the similar neighbors within pre-defined threshold τ. We defer the discussion to Section IV-B.

With the retrieved similar neighbors, the class membership of the query e_q can be inferred by the join operation of the neighbor set $\mathcal{N}(e_q)$ with $Human\_Labels$ table to obtain its associated list of classes and corresponding decisions. In a multi-label problem, one example of the join result is the following table.

entity	class	dec.
e_1	c_1	T
e_2	c_1	T
e_3	c_1	F
e_3	c_2	F

For the query entity e_q's class membership, we can learn its possible classes and decisions by grouping the attribute $class\_id$ (class) to collect the $decision$ set (dec.). One simple way to handle the conflicting $decision$ values is taking only decisions that are unanimous among the neighbors, which is effective in preventing propagating incorrect labels. We adopt this strategy in order to exclude the labeling noise resulting from human annotation errors. Now, after resolving conflicting decisions from similar neighbors, we do not make any prediction on the class c_1 for query entity e_q due to a non-unanimous set of neighbor's decisions such as $\{T, T, F\}$. As a result, our model will only predict e_q as it is not a member of class c_2 based on its neighbor e_3 record (e_3, c_2, F). We infer the decision l_2^q of e_q w.r.t. class c_2^q is $False$.

In this problem, the queries can be posed in the class definition stage when the class assignment is tentative or in the post-classification audit phase to alleviate the concept drift. Given a query-class pair (e_q, c), the classification reviser will generate two types of prediction for the decision l_c^q: **is-member** and **is-not-member** (corresponding to $flag = True$ and $False$ in Problem 1, respectively). If the neighbors $\mathcal{N}(e_q)$ make inference on the class c as *true* with high confidence score, it is considered as an is-member prediction. Similarly, if the neighbors $\mathcal{N}(q)$ make prediction on the class c as *false* with a high confidence score, we call it a is-not-member prediction.

To determine how likely a query and class decision $(e_q, c, flag)$ is a revision candidate, we define the revision confidence score as:

$$s_{(e_q,c)} = \sum_{e_p \in \mathcal{N}(e_q)} w(dist(v_p, v_q)) \qquad (2)$$

where $w()$ is an anti-monotone weight function that takes the distance of two embeddings v_p and v_q from similar entities e_p and e_q. A higher score implies a higher prediction confidence. We let the weight function give weights to different distance bands. Thus, more neighbors in making the decision and/or closer embedding distances (i.e. higher quality neighbors) enable the model to have a better confidence. We generate a ***revision signal*** when our prediction on a class c for e_q is not aligned with e_q's current decision l_c^q in the $Inferred\_Labels$

[2]Simply associating the set of true properties to each entity is not practical in the real-world classification systems.

table and its confidence score is $s \geq t$, where t is a pre-defined score threshold.

IV. Classification Revision Method: NNCR

The workflow of the method is illustrated in Figure 1. The goal of a classification revision model is to make a prediction for the decision l on all classes $C = \{c_1, c_2, \cdots, c_L\}$ of a high label cardinality $|L|$ for all input queries $Q = \{q_1, q_2, \cdots q_m\}$ by using human labeled data (i.e., records in $Human\_Labels$) $I = \{i_1, i_2, \cdots i_n\}$. In practice, $|L|$ can be in hundreds of thousands, and m and n may be very large in the order of hundreds of millions where $m >> n$.

One common use case is that batch queries come towards the incremental database frequently. The job of the classification reviser is to finish inference in a few hours – revising classifications of new items needs to be performed efficiently and timely.

We do this by leveraging an efficient and scalable nearest neighbor search which finds top similar entities with known labels in a very large high-dimensional embedding database. The hypothesis is that similar entities have similar class assignments and hence human-audited labels can be propagated to query entities.

In order to obtain similar neighbors $\mathcal{N}(e_q)$ for query entity e_q in Q, we first learn an entity representation so that feature vectors taken as the distance function inputs can be used to define the similarity. A recent work [12] shows that KNN method combined with a strong off-the-shelf generic feature extractor outperforms the state of the art algorithm on a particular problem. We leverage pre-trained embeddings to represent entities, as they are popular, can be pre-computed, and fast in inference. We discuss the embeddings in Section IV-A.

In Section IV-B, we describe the technical details of how we employ a GPU powered state-of-the-art model [14] to distribute hundreds of millions of query embeddings and perform exact search for similar neighbors from an indexed labeled data set. The ideas of NNCR can be extended to approximate nearest neighbor search algorithms [31] such as Locality-Sensitive Hashing (LSH) [32], HNSW [33] and scaNN [34], in applications where some accuracy loss is acceptable. Indeed, we are more interested in examining how much more accuracy we can enjoy from the exact search.

A. Feature Embedding

As an application example—which is also what we do in our experimental evaluation—we describe, in this subsection, how item embedding can be performed for revising multi-label classifications in product directories. In a product directory, each entity is characterized by multiple modalities with text attributes (such as product name, brand, manufacturer, general keywords, and bullet points), major images, and some structural relationships.

Single vs. multiple embeddings. The design choices include what features to encode in the embedding and the loss functions. Each feature can be learned separately or jointly. One can use weighted aggregation of multiple embeddings learned from different models into one final entity representation and use it for the similarity search. Although this allows a more lightweight system, we choose to have a single embedding (e.g. image-based embedding or text-based embedding) used in computing similar neighbors. This produces better interpretability in diagnosing anomalies.

We choose to encode an entity title as a feature vector of dimension 128 to represent each entity. Entity title is the most critical field as it describes the product concisely and usually contains brand, manufacturer and keywords as very useful information. Its dimensionality is typically smaller compared to image and multi-modal embeddings, which is better from both scalability and cost perspectives. However, our neighbor-based similarity search method is not limited to this, and can work with other pre-trained models such as image-based embeddings [35] and embeddings based on early/late fusion in vision and language modeling [36]–[38] as well.

One attraction of the Word2Vec [39] algorithm is its scalability, which comes from the negative sampling as an approximation of the softmax loss over the space of vocabulary. The skip-gram architecture is used to vectorize textual inputs. Then, we rely on the TF-IDF weighted average on the Word2Vec token representations. First, a skip-gram model is trained on 143 million product titles and descriptions. The implementation adopts Gensim[3] with following parameters: embedding size = 128, maximum distance between current and predicted word (i.e. window) = 5, minimum frequency = 10, number of workers = 60, negative samples = 20 to learn the word vectors. The term frequency (tf) and inverse document frequency (idf) are calculated on the same training data, where each document is a product title and description. We use the weighted average to generate the weighted embedding as entity embedding v_e where we consider all words but ignore the contribution of unknown words:

$$v_e = \frac{\sum_{w!='UNK', w \in V} (tf_w \times idf_w \times vec[w])}{\sum_{w!='UNK', w \in V} tf_w \times idf_w} \tag{3}$$

where w are tokens of product title, 'UNK' is unknown word, V is the vocabulary, $vec[w]$ is the word vector of word w. An alternative is to use deep pre-trained (large) language models such as BERT [40], XLNet [41], RoBERTa [42] and OpenLLaMA [43], however, the fine-tuning procedure of transformer-based models on large-scale label space usually requires lengthy training time and memory consumption even with powerful GPUs. Thus, the Word2Vec vectorization remains a competitive solution for hundreds of millions of data items.

B. Similar Neighbors Search

Here, we call the labeled data as the *indexed set*. The overall algorithm for the embedding-based KNN search problem is in Algorithm 1.

[3]https://radimrehurek.com/gensim/models/word2vec.html

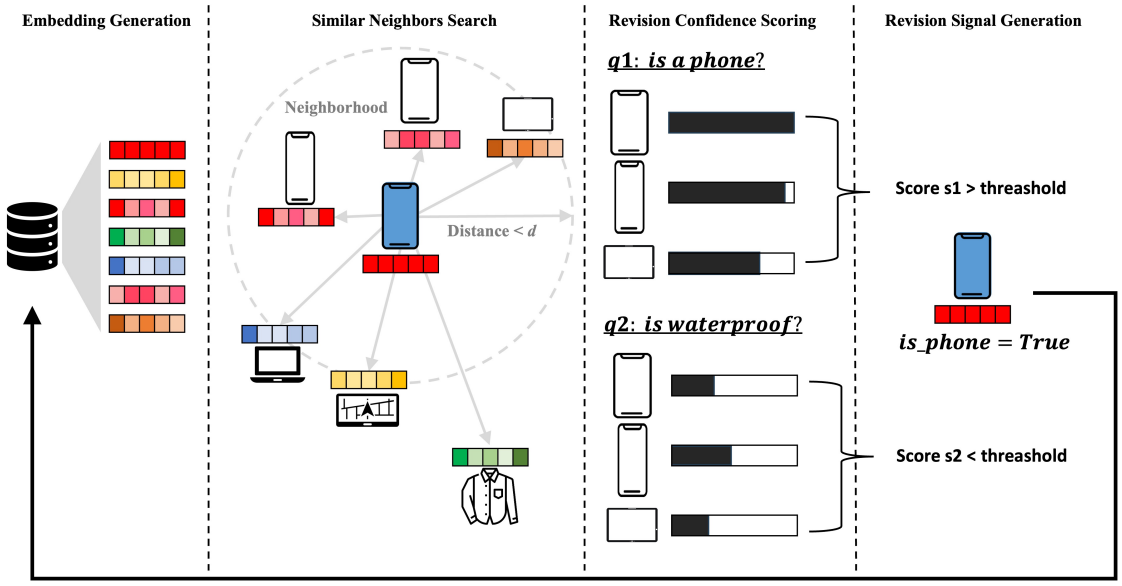


Fig. 1. A product directory example of NNCR overview. (1) The Embedding Generation: vectorize all the items in the database and in the query queue. (2) Similar Neighbors Search: retrieve neighbors for the query items, e.g. at the center of the dotted circle. (3) Revision Confidence Scoring: for each query item and label pair, it aggregates the neighbors and computes the weighted sum based on distances as the confidence score. (4) Revision Signal Generation: generate the high-strength signals, e.g. `is_phone=True`, when the score is above a threshold and the value, e.g. `True`, is not consistent with the one currently stored in the database, e.g. `None`.

Given a set of input queries Q of the form $q = (e_q, c_i^q, flag)$, we take the distinct set of query entities e_q and get its embedding vector v_q. Then, find the neighbors for all query entities by Algorithm 1. For the purpose of simplicity and clarity, we use the following renaming in this section. We let query data q_m be (e_m, v_m) and indexed data i_n be (e_n, v_n).

First, we randomly partition the indexed set $I = \{i_1, \cdots, i_n\}$ into y chunks and build the index for the associated embeddings (lines 2 – 3 in Algorithm 1). Next, we randomly partition $Q = \{q_1, \cdots q_m\}$ into x shards and distribute each shard Q_i into different instances where each shard has one of the index replicas. For the query embedding in shard Q_i, we load the index for the embedding vectors from each chunk I_j, so both embedding sets can fit in the memory of a multi-core GPU machine. Since we have y chunks in I, given the target number of neighbors k, we retrieve and merge k' neighbors from each index chunk (e.g., $k' = k/y$).

For each query shard Q_i, we find the top k' similar neighbors from each chunk I_j of the indexed set (line 7) to get a neighbor set $\mathcal{N}(e_m)$ for all query entities in Q_i. However, it is possible that a query entity e_m and its similar neighbor e_n retrieved from $\mathcal{N}(e_m)$ are actually still far apart in the embedding space. Thus, we use a pre-defined distance threshold τ to post-process similar pairs by filtering out pairs whose L_2 distance is greater than τ. Hence, we get the top $k^\star$ neighbors from each chunk for a query entity. After applying the distance filter, we get the similar pairs $I_j.pairs^*$ from chunk I_j for queries in shard Q_i (line 8). Finally, we merge top $k^\star$ neighbors from y chunks for all query entities in Q_i to get similar pairs $Q_i.pairs$ (line 9). We collect similar pairs $Q_i.pairs$ from x shards as our output similar pairs.

Algorithm 1: Similar Neighbors Search

Input: Indexed set I, Query set Q, threshold τ, max # neighbors k', x shards, y chunks

1 Partition(I, y), Partition(Q, x)
`/* Build index */`
2 **for** *each chunk* I_j *in* I **do**
3 Add I_j into index
`/* Search */`
4 **for** *every query shard* Q_i *in* Q **do**
5 **for** *each chunk* I_j *in* I **do**
6 Load I_j into memory
7 $I_j.pairs \leftarrow$FindNeighbor(Q_i, I_j, k')
8 $I_j.pairs^* \leftarrow$ FilterPairs($I_j.pairs$, τ)
9 $Q_i.pairs \leftarrow$ Merge($\{I_1.pairs^*, \cdots, I_y.pairs^*\}$)
10 **return** $\{Q_i.pairs : Q_i \in Q\}$

V. Experiments

In this section, we present a comprehensive performance study of the proposed method on large-scale product data. We ran the similarity search algorithm on an AWS ml.p3.16xlarge instance (8 Nvidia V100 GPUs and 128 GB RAM). The rest of the experiments were conducted using Apache Spark on an AWS EMR cluster with 40 r4.8xlarge AWS instances. We aim to answer the following research questions (RQ):

- **RQ1**: How does NNCR perform compared to other state-of-the-art methods?
- **RQ2**: Which KNN/ANN algorithms have a robust implementation that scales to hundreds of millions of queries

and indexed vectors (of 128 dimensions)?
- **RQ3**: What are the qualities of the embeddings and the returned query-neighbor pairs?
- **RQ4**: What are the best parameters for online settings? What is the online performance gain as we use the exact search?

A. Data

1) Datasets:

- **Random-split-based data**. We sample data from our labeled store on recent snapshots. This dataset contains 114 million products over around 73,000 classes where the class distribution is long tailed and skewed. We use a 70%-30% split to obtain randomly sampled training/testing sets. We take the testing set as batch queries and search neighbors from the indexed training set. We use such a dataset to evaluate our method and to compare against baseline methods, conducting a systematic analysis on neighbors and embeddings as it reflects a long-tailed class distribution in the query data. However, since 30% of the labeled data is left as queries, the parameters, cost budget and running times computed on this dataset does not give us the best estimation for large-scale online settings. Thus, we collect another dataset by a training-testing split over the time dimension, where the parameters selected are suitable for an online environment.
- **Time-based data**. For the training/testing dataset split over the time dimension, we use all the labeled data from a recent snapshot as the training set and collect the new data coming to the queue from the subsequent time period as our testing set. These queries could be associated with new classes or some existing classes.

2) Data Preprocessing Optimization: To reduce the search space and running time, we cluster the entities with close embeddings (i.e. embedding distance is less than a small value ϵ that is close to 0) into a group where each group has a unique id (i.e. representative entity id) and a representative embedding. This compression reduces the search space by 35% and saves the redundant computation in similar neighbor search. We group all the query entities $e_q \in Q$ with almost the same embedding. We repeat the same *GroupBy* operation to get the group id for each indexed entity $e_i \in I$ and choose one embedding from a group as the representative embedding for each group. The similar neighbors search (Algorithm 1) is performed on representative space to get the similar pairs at group level. The total similar pairs on original entity space can be mapped back by the cartesian product of two groups group(e_q) $\times$ group(e_i) where group(e_q) (or group(e_i)) contains all entities with close embedding distances to entity e_q (or e_i).

B. Baselines

We evaluate NNCR against the following state-of-the-art approximate nearest neighbor algorithms.

- **`NNCR-ivf`**. We follow our Algorithm 1 for indexing and searching, but replace the actual index by the approximate version FAISS-ivf [14]. The best parameters learnt from our dataset are $nlist = 25$, $probe = 1$, $k = 30$.
- **`LSH`**. A recent review paper [44] revisits the problem of search with locality sensitive hashing (LSH) and makes a thorough comparison of eleven popular hashing algorithms, the random-projection-based LSH [45] ranks first despite its simplicity. This method is implemented in spark.ml with AND amplification using parameters: $bucketLength = 0.1$, $numHashTables = 6$. With the AND amplification, two entities are considered as candidate neighbors if and only if the projected hash values are identical.

TABLE I
MULTI-LABEL PREDICTION PERFORMANCE. ABSOLUTE IMPROVEMENT OVER BASELINE FOR PREC. AND REC.

Method	Prec.	Rec.	Volume
NNCR 15	1.1%	**21.6%**	208,495,941
NNCR 30	1.3%	20.7%	301,223,481
NNCR 100	**2.0%**	17.4%	557,697,189
NNCR 150	**2.0%**	16.0%	**680,009,787**
NNCR-ivf	1.3%	19.0%	301,908,452
Baseline LSH	-	-	305,577,294

C. Performance Comparison (RQ1)

We compare NNCR against the baselines on the Random-split-based data dataset using several offline metrics as detailed below.

Offline evaluation metrics. The primary metrics we use for the offline evaluation are:

- **Precision**. Fraction of predictions on queries $q = (e_q, c_i^q, flag)$ that are correct, i.e., $Precision = \frac{TP}{TP+FP}$, where TP, FP are the numbers of true and false positives, respectively.
- **Recall**. Fraction of ground truth queries $q = (e_q, c_i^q, flag)$ that are predicted correctly, i.e., $Recall = \frac{TP}{TP+FN}$, where FN is the number of false negatives.
- **Prediction Volume**. The total number of predicted statements $q = (e_q, c_i^q, l_i^q)$ that can be made over all entities in the input query set.

In Table I, NNCR outperforms the baselines in precision, recall and prediction volume as expected. We can see that NNCR gives a lower precision and a higher recall for small range of k (i.e. $k = 15, 30$) as compared to large range of k (i.e. $k = 100, 150$). In terms of the prediction volume, NNCR is able to generate more predictions on (query, class, flag) triples with a larger k. However, not all the predictions have the ground truth label in the data so it does not imply a higher recall. The average number of neighbors retrieved from LSH is around 100. It is noteworthy that, with much fewer neighbors (i.e. $k = 30$), the prediction volume that NNCR can produce is in the same ballpark as LSH. We can see that $k = 15$ gives the best results in terms of F1 score. However, this is not our final parameters as 30% of this indexed data was left out as testing set for evaluation. The message from this experiment

is that we can reduce the search time as using fewer neighbors in NNCR outperforms LSH with a larger neighbor set.

D. Scalability Analysis (RQ2)

Data preprocessing effectiveness. With the results in Table II, we demonstrate that the similarity search on representative embeddings is effective in very large scales. We compare the running time, total number of similar entity pairs (on the original entity space), and the percentage of retrieved similar pairs being filtered out. We start with a range of $k \in [80, 320]$ but we do not go beyond the the range when the job cannot finish within reasonable hours. The running time in representative space can be reduced by 25%. Also, we are interested in what fraction of returned neighboring pairs is actually similar in terms of the embedding features, i.e. not being filtered out by the pre-defined distance threshold τ, as we do not want to waste computation on retrieving dissimilar neighbors. The filtered percentage increases as k goes larger, and it does not converge within our target execution time. Also, with the same value of k, the total number of similar pairs obtained from mapping the representative entity back to the original entity is significantly larger than the similar pairs found by the search on the original entity embedding. More similar pairs are preferred and critical in our application as these pairs contribute to the volume of classification revisions.

Ruling out `NNCR-ivf`. When considering ANN algorithms, NNCR-ivf outperforms LSH in terms of the precision and recall. However, the total inference time of LSH is 1.3 hours and is the fastest and most cost-effective solution as compared to NNCR and NNCR-ivf. Because of its performance stability concern[4], NNCR-ivf is not considered further.

Next, we present the total running time of NNCR used for building the index and searching for top k neighbors. The average of 5 runs of total indexing time is 29.4 seconds. The search time on average is 9259.3 seconds.

E. Qualitative Embedding Analysis (RQ3)

Embedding distance threshold. From the pre-trained embedding, we empirically determine the distance threshold τ as 1.0 based on embedding analysis. The parameter τ is embedding-dependent and the distance metric used in this similarity definition is L_2 distance. In Table III, with distance in the range of [0, 0.3), we consider the pairs as being very similar. We see that both products on the left and right are about the Scooter. With the distance 0.2, they share the same brand name but differ in color. The next range is [0.3, 0.5). With a distance very close to 0.5, we can see that both product names are similar except for the manufacturer-related text in the second row of Table III. The last distance range band is close to the threshold cutoff. With the distance 1.0, the bottom example in Table III shows that both products look differently but they share many critical fields such as stores, materials, styles, safety properties, functionalities, and manufacturer's origins.

Neighbor distance distribution. We look into the similar pairs returned by NNCR and LSH to see if more unique similar pairs can be identified with NNCR. The neighbor pairs can be categorized into 3 sets: S_k – the similar pairs caught by NNCR, S_a – the similar pairs caught by LSH, $S_k \cap S_a$ – the overlapping pairs found by both NNCR and LSH. We compute the Jaccard Similarity ($Jaccard(S_k, S_a) = \frac{|S_k \cap S_a|}{|S_k \cup S_a|}$) between the sets S_k and S_a w.r.t different similar pairs retrieved by different k, which measures the similarity between the two sets. Also, for all the similar pairs found by NNCR, we calculate the fractions of overlapping pairs it has with S_a. In Table IV, the percentage of overlap pairs in S_k is low and this number drops with a larger k. A similar trend can be seen in the Jaccard similarity index. The Jaccard similarity is in the range [0.06, 0.11] which is close to 0, so we conclude that the pairs found by both methods are very different and potentially detect different revision candidates in the data.

Next, we focus on the quality of these non-overlapping similar pairs: $S_k \setminus S_a$ – the similar pairs caught only by NNCR and $S_a \setminus S_k$ – the similar pairs caught only by LSH. As shown in Table V, we present the distribution of the embedding distance between queries and neighbors retrieved by each method. In the low and medium embedding distance range $[0, 0.3)$ and $[0.3, 0.5)$, we observe more similar pairs from $S_k \setminus S_a$ as compared to $S_a \setminus S_k$, where these range buckets are mapped to a higher weight band in computing the revision confidence score. By contrast, a larger fraction of similar pairs from $S_a \setminus S_k$ is around the higher embedding distance range $[0.5, 1.0]$. Therefore, the similar pairs generated by NNCR result in a better quality neighbor set, which in turn produce higher confidence in classifications revision tasks.

F. Parameters Selection for Online Settings (RQ4)

In this section, we discuss the parameter selection for an online setup. The experiments are done on the Time-based data where it reflects the actual number of embedding vectors we have to index in memory and is appropriate for selecting the number of neighbors k. We first select a few candidate parameters by using the offline evaluation metrics precision and recall. Then, the final parameter k and revision confidence score s are determined by the running time, cost budget, online metrics, and its additional value to an ensemble model served in a multi-label classification revision system.

Even though NNCR can scale out easily, we explore k in the small-value range as the cost budget on the total number of GPU instances is factored in. We select the candidate k and s based on precision and recall. The best parameters occur at $k = 15, s \geq 0$ in terms of precision and recall in Table VI, which is consistent with the best results we obtain from Random-split-based data in Table I. When k increases, as we define the neighbors by embedding threshold τ which ensures the returned neighbors are indeed similar, the precision improves because any label decision is agreed on by more neighbors. On the other hand, with larger k, we observe that

[4]We notice that NNCR-ivf sometimes runs into CUDA errors, potentially due to certain combination of the number of indexed vectors and query vectors.

TABLE II
SEARCH RESULTS ON REPRESENTATIVE VS. ORIGINAL EMBEDDING

	Representative Space				Original Space		
k	time (hr)	total rep. pairs	filtered %	total pairs	time (hr)	filtered %	total pairs
80	**3.1**	759,631,279	24.0	**3,139,190,535**	4.2	19.7	1,046,375,190
160	**4.5**	1,407,763,576	29.5	**4,686,662,804**	5.2	24.9	1,957,768,228
320	**5.7**	2,003,991,405	33.0	**6,074,264,607**	7.3	30.9	3,603,563,716

TABLE III
EMBEDDING EXAMPLES

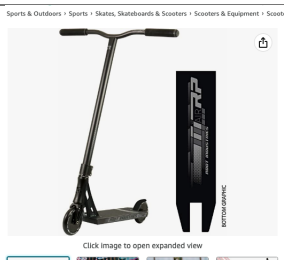
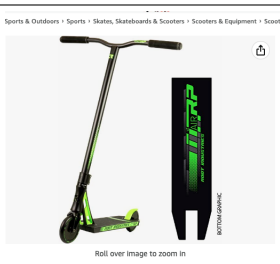
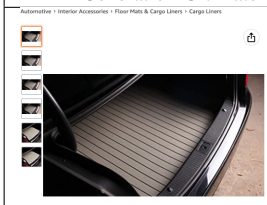
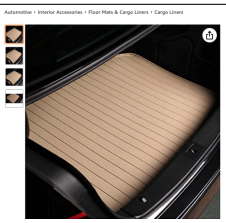
Embedding Distance = 0.2	
ROOT INDUSTRIES AIR RP Complete Scooter Green	ROOT INDUSTRIES Air Rp Scooter Black
	
Embedding Distance = 0.5	
AudelTech Custom Fit Jeep Wrangler Cargo Liner Next Generation Ultimate Luxury Cargo Area Tray Liner	AudelTech Custom Fit Chevrolet Malibu Cargo Liner Next Generation Ultimate Luxury Cargo Area Tray Liner
	
Embedding Distance = 1.0	
Bleacher Creatures NFL Buffalo Bills Kids Jim Kelly Plush Figure, 10", Blue	Bleacher Creatures MLB Unisex MLB 10-inch Mascot Plush Figure
	

TABLE IV
NEIGHBOR SET COMPARISON

k	$\lvert S_k \cap S_a \rvert$	$\lvert S_k \setminus S_a \rvert$	overlap	$\lvert S_a \setminus S_k \rvert$	Jac.
80	443M	2,695M	14.1%	970M	0.11
160	481M	4,205M	10.2%	932M	0.09
240	516M	5,558M	8.4%	897M	0.07
320	537M	6,795M	7.3%	876M	0.06

recall drops. This is because currently we adopt the unanimous decision in aggregating neighbors, we do not make inference on a class for a query entity under the condition that the neighbors have conflicting decisions. Thus, it increases the chance of false negatives because decisions proposed by close neighbors can be opposed by farther neighbors. However, it is also possible that, with larger k, the recall improves. This could result from the predictions exclusively contributed by farther neighbors. Hence, we consider both $k = 15$ and $k = 30$ as good candidates and use the online metrics to determine the final parameters.

In an online setting, we run experiments to use the labeled data and make inference on the incoming query-class pairs. Due to the label unavailability of these new queries, we use the online metrics to estimate its performance and compare against a proprietary model.

Method for comparison. We compare against a proprietary

TABLE V
NEIGHBOR EMBEDDING DISTANCE DISTRIBUTION

	$S_k \setminus S_a$			$S_a \setminus S_k$		
dist. range	$[0, 0.3)$	$[0.3, 0.5)$	$[0.5, 1.0]$	$[0, 0.3)$	$[0.3, 0.5)$	$[0.5, 1.0]$
$k = 80$	**0.11**	**0.20**	0.68	0.03	0.06	**0.91**
$k = 160$	**0.08**	**0.15**	0.78	0.02	0.06	**0.92**
$k = 240$	**0.06**	**0.12**	0.82	0.02	0.05	**0.92**
$k = 320$	**0.05**	**0.11**	0.85	0.02	0.05	**0.93**

TABLE VI
PARAMETERS SELECTION. ABSOLUTE IMPROVEMENT OVER BASELINE.

score	$s \geq 0$		$s \geq 1$		$s \geq 2$		$s \geq 3$	
k	prec.	rec.	prec.	rec.	prec.	rec.	prec.	rec.
15	**1.0%**	**8.8%**	**0.0%**	9.0%	**0.0%**	6.5%	**0.1%**	5.0%
30	**1.0%**	3.0%	**0.0%**	3.0%	**0.0%**	2.0%	**0.0%**	1.0%
Baseline 40	-	-	-	-	-	-	-	-

TABLE VII
PREDICTION PERFORMANCE COMPARISON

method	Signals	is-mem.	non-mem.
NNCR 15	13,584,664	8,929,395	4,655,269
NNCR 30	**16,002,255**	**10,735,293**	**5,266,962**
LSH	5,111,315	3,601,848	1,509,467

model that is an ensemble [15] of several state-of-the-art algorithms in this domain such as a gradient tree boosting method XGBoost [46] and an extreme multilabel ranking method PECOS [7].

Online evaluation metrics. Besides the expected precision and recall, we look for the expected number of revision candidates that NNCR can identify.

- **Expected signals**. Given a set of query-class pairs, the total number of signals, including is-member prediction and non-member prediction with confidence score s greater than or equal to a given threshold.
- **Uniqueness**. The number of unique signals yielded by NNCR but not being covered by any of the algorithms included in an ensemble model. This measure gives the idea of the additional value or gain that a model can complement to the overall performance of a multi-label classification revision system.

Results. Table VII shows the performance comparison between two candidate parameters and the baseline LSH. The expected number of signals from NNCR with $k = 15$ (resp., $k = 30$) is 2.7 (resp., 3.1) times more than that from LSH. Both k values give significantly more potential is-member predictions and is-not-member predictions. We expect that, with a larger k, the number of signals increases as this raises the chance for more neighbors to propose diverse signals. Since both approaches use the same embedding, we look at the overlap of the generated signals. Out of all 13M signals suggested by $k = 15$, only 21% of them overlap with the signals generated by LSH. With $k = 30$, the fraction of overlap signals can drop down to 19%, demonstrating that more potential revision candidates can be identified in addition to the ones also caught by LSH.

Ensemble model comparison. We further look into the uniqueness of NNCR signals as compared to signals produced by an ensemble model. For $k = 15$, the overlap is 8,947,848 and the unique number of signals is 4,636,816. Thus, from all signals of NNCR, 34.1% of the signals are unique. We consider this 34.1% as being very competitive because the ensemble model relies on several data modalities and algorithms.

Among these overlap signals, 21% of them are overlap with strong is-member prediction and 13% of them are overlap with strong non-member prediction and so the rest 66% of NNCR signals can potentially help to surface out more revision candidates as it gives high quality signals.

The best number of signals occurs at $k = 30$ although this introduces a longer running time from 2.6 hours to 3.5 hours per GPU instance. The number of expected signals and the unique signals increased as expected. Out of all 16M signals, still 34% of NNCR signals are unique as compared to all signals from ensemble model. Thus, we choose $k = 30$ as its competitive performance on volume of unique signals and expected precision as compared with its $k = 15$ counterpart. In general, we see that NNCR powered by GPU optimization is a good choice for multi-label classification revision problem due to its high precision and recall. When some accuracy can be traded off, LSH with pre-trained embedding is also a frugal solution.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel classification reviser for a large-scale data, which is a KNN based method with neural embeddings, namely NNCR. We present the internal details of NNCR, from feature vectorization to the similar-neighbor search algorithm for classification revision, leveraging the state-of-the-art GPU-optimized neighbor-search indexing method. Our comprehensive experiments on large product data demonstrate the effectiveness of NNCR, as well as determine the best parameter choice for online settings.

As future work, we plan to examine more advanced embedding methods as part of metric learning in connection to multi-label classifications.

VII. Acknowledgments

We are grateful for the helpful comments provided by Andrew Borthwick, Ismail Tutar and Gang Luo.

References

[1] M. Wydmuch, K. Jasinska, M. Kuznetsov, R. Busa-Fekete, and K. Dembczynski, "A no-regret generalization of hierarchical softmax to extreme multi-label classification," *NeurIPS*, 2018.

[2] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[3] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[4] Y. Tagami, "Annexml: Approximate nearest neighbor search for extreme multi-label classification," in *KDD*, 2017.

[5] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma, "Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising," in *WWW*, 2018.

[6] 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," *NeurIPS*, 2019.

[7] H.-F. Yu, K. Zhong, J. Zhang, W.-C. Chang, and I. S. Dhillon, "Pecos: Prediction for enormous and correlated output spaces," *JMLR*, 2022.

[8] T. Jiang, D. Wang, L. Sun, H. Yang, Z. Zhao, and F. Zhuang, "Lightxml: Transformer with dynamic negative sampling for high-performance extreme multi-label text classification," in *AAAI*, 2021.

[9] W. Zhang, J. Yan, X. Wang, and H. Zha, "Deep extreme multi-label learning," in *Proceedings of the 2018 ACM on international conference on multimedia retrieval*, 2018, pp. 100–107.

[10] J. Liu, W.-C. Chang, Y. Wu, and Y. Yang, "Deep learning for extreme multi-label text classification," in *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, 2017, pp. 115–124.

[11] Y.-H. Kuo, Z. Li, and D. Kifer, "Detecting outliers in data with correlated measures," in *CIKM*, 2018.

[12] L. Bergman, N. Cohen, and Y. Hoshen, "Deep nearest neighbor anomaly detection," *arXiv preprint*, 2020.

[13] M. Zhao, J. Chen, and Y. Li, "A review of anomaly detection techniques based on nearest neighbor," in *CMSA*, 2018.

[14] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, 2019.

[15] L. Rokach, "Ensemble-based classifiers," *Artificial intelligence review*, vol. 33, no. 1, pp. 1–39, 2010.

[16] C. C. Aggarwal, "An introduction to outlier analysis," in *Outlier analysis*. Springer, 2017, pp. 1–34.

[17] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," ser. SIGMOD, 2000.

[18] E. M. Knorr and R. T. Ng, "Algorithms for mining distance-based outliers in large datasets," in *VLDB*, 1998.

[19] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *SIGMOD*, 2000.

[20] M. Sugiyama and K. Borgwardt, "Rapid distance-based outlier detection via sampling," in *NeurIPS*, 2013.

[21] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *ICDM*. IEEE, 2008, pp. 413–422.

[22] T. T. Dang, H. Y. Ngan, and W. Liu, "Distance-based k-nearest neighbors outlier detection method in large-scale traffic data," in *2015 IEEE International Conference on Digital Signal Processing (DSP)*, 2015, pp. 507–510.

[23] H. Xu, L. Zhang, P. Li, and F. Zhu, "Outlier detection algorithm based on k-nearest neighbors-local outlier factor," *Journal of Algorithms & Computational Technology*, vol. 16, p. 17483026221078111, 2022.

[24] L. Wang, J. Li, U. A. Bhatti, and Y. Liu, "Anomaly detection in wireless sensor networks based on knn," in *ICAIS*, 2019.

[25] B. Wang, S. Ying, G. Cheng, R. Wang, Z. Yang, and B. Dong, "Log-based anomaly detection with the improved k-nearest neighbor," *IJSEKE*, 2020.

[26] Y. Liao and V. R. Vemuri, "Use of k-nearest neighbor classifier for intrusion detection," *Computers & security*, 2002.

[27] J. T. Andrews, E. J. Morton, and L. D. Griffin, "Detecting anomalous data using auto-encoders," *IJMLC*, 2016.

[28] T. Ergen and S. S. Kozat, "Unsupervised anomaly detection with lstm neural networks," *IEEE transactions on neural networks and learning systems*, 2019.

[29] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly detection using one-class neural networks," *arXiv preprint*, 2018.

[30] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *ICML*, 2018.

[31] M. Aumüller, E. Bernhardsson, and A. Faithfull, "Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," in *International conference on similarity search and applications*, 2017.

[32] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive data sets*. Cambridge univ. press, 2020.

[33] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *TPAMI*, 2018.

[34] R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar, "Accelerating large-scale inference with anisotropic vector quantization," in *International Conference on Machine Learning*, 2020. [Online]. Available: https://arxiv.org/abs/1908.10396

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[36] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *ICML*, 2021.

[37] W. Kim, B. Son, and I. Kim, "Vilt: Vision-and-language transformer without convolution or region supervision," in *ICML*, 2021.

[38] J. Li, R. Selvaraju, A. Gotmare, S. Joty, C. Xiong, and S. C. H. Hoi, "Align before fuse: Vision and language representation learning with momentum distillation," *NeurIPS*, 2021.

[39] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *NeurIPS*, 2013.

[40] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint*, 2018.

[41] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *NeurIPS*, 2019.

[42] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint*, 2019.

[43] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[44] D. Cai, "A revisit of hashing algorithms for approximate nearest neighbor search," *TKDE*, 2019.

[45] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *ACM symposium on Theory of computing*, 2002.

[46] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *KDD*, 2016.