Reference Product Search

Chu Wang*
Amazon.com
chuwang@amazon.com

Lei Tang†
Amazon.com
leitang@amazon.com

Shujun Bian
Amazon.com
sjbian@amazon.com

Da Zhang
Amazon.com
dazh@amazon.com

Zuohua Zhang
Amazon.com
zhzhang@amazon.com

Yongning Wu
Amazon.com
yongning@amazon.com

ABSTRACT

For a product of interest, we propose a search method to surface a set of reference products. The reference products can be used as candidates to support downstream modeling tasks and business applications. The search method consists of product representation learning and fingerprint-type vector searching. The product catalog information is transformed into a high-quality embedding of low dimensions via a novel attention auto-encoder neural network, and the embedding is further coupled with a binary encoding vector for fast retrieval. We conduct extensive experiments to evaluate the proposed method, and compare it with peer services to demonstrate its advantage in terms of search return rate and precision.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; **Search methodologies**; **Machine learning**.

KEYWORDS

Product Search, Representation Learning, Semantic Hashing, Attention Mechanism, Denoising Auto-Encoder

ACM Reference Format:
Chu Wang, Lei Tang, Shujun Bian, Da Zhang, Zuohua Zhang, and Yongning Wu. 2019. Reference Product Search. In *Companion Proceedings of the 2019 World Wide Web Conference (WWW '19 Companion), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3308560.3316598

1 INTRODUCTION

Product modeling and retrieval tasks, including product advertising, substitution, and recommendation, are fundamental for e-commerce business. To help customer discover more products and improve shopping experience, e-commerce platforms like Amazon, eBay, Taobao, and JD all gradually launched new browsing or assistance features related to similar product comparison, alternative product recommendation, and substitute product suggestion. In addition to the traditional query-to-product search, such a product-to-product retrieval mechanism contributes significantly to the e-commerce business. For example, JD launched a "find similar" widget available for certain products, and Amazon also provides this widget for products in the browse history, to provide more alternatives closely related to a product of customers' interest. In general, the solutions consist of two stages: a candidate product set is retrieved first, followed by a task-specific ranking model to generate the results. Often, research interests focus on a ranking model built to optimize towards such a business application, but a suitable candidate product set is required to feed the ranking model and it is not well discussed in the literature.

Qualification of the candidate product set differs for different business objectives. In spite of those differences, the candidate product set is generally sourced from catalog information, behavioral data, and human annotations. The traditional inverted-index based retrieval relies on indices which are generated via product attribute tagging, and products sharing common indices like keywords are regarded as the candidates [12, 28]. If the products are assigned or classified in a taxonomy, the products under the same category can be used as the candidates, though mis-classified query product will lead to irrelevant results, and the size of the retrieved candidate set is uncontrollable and may be highly skewed. Customer behavioral information is another source to generate candidate products. Products that often viewed together or purchased together can serve as candidates for each other [19]. Though widely adopted in e-commerce business especially for product advertising and recommendation, the mentioned methods need to handle issues like feature sparsity, noisy data sources, low return rate, and inability to adjust the retrieval size. Ideally, the retrieval approach should be able to fetch a enough number of candidate products similar to the query, and the number of candidates should be flexible to adjust in order to accommodate various downstream ranking models for e-commerce applications.

In this paper, we focus on the problem of obtaining a set of reference products for a query product of interest. The reference products can be further fed to a ranking or relevance model to optimize the business objectives like clicks, conversions, or profits. We formulate this product retrieval problem as a search task, where we build product embedding vectors, quantify product similarity by vector distance, and conduct nearest neighbor search (NNS) in the vector space to surface the reference products. In addition to the requirement of flexibly adjusting retrieval size that NNS naturally provides, we focus on two metrics to evaluate the quality of the proposed search method. The precision of the search results measures

*† Both authors contributed equally.

WWW '19 Companion, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6675-5/19/05.
https://doi.org/10.1145/3308560.3316598

the quality of the top search results; the return rate estimates the ability to retrieve enough reference products for different queries.

There are various challenges involved to build a desired product-to-product search method. Customer behavior data is limited to only popular products, making it difficult to achieve high return rate. The catalog information is more widely available for product embedding, but feature extraction from low-quality or even missing product information is challenging [5, 36]. In fact, we observe 10% to 45% missing rates for several important product fields, and the available catalog information is plagued by poorly written catalog data with irrelevant or duplicated information. Large-scale vector search is also challenging in terms of the computation efficiency. Exact NNS is not realistic for moderately large datasets [33], and one need to refer to approximate methods in order to reduce the latency. For datasets with a large volume of items, the balance between the efficiency gain and the quality loss is difficult to handle.

In this paper, we propose a novel attention auto-encoder neural network to build high-quality and robust product vectors. In order to achieve better search precision and return rate, the negative impact of missing or low-quality attributes is minimized via attention mechanism [29]. The vector search is then conducted by an optimized semantic hashing algorithm [23], where each product embedding is coupled with a binary encoding so that NNS can be achieved by searching in the vicinity of the query encoding. The encoder is optimized to penalize tiny and huge binary buckets for better search precision and latency. Compared to existing product-to-product retrieval methods, the proposed method is able to obtain better precision while enhancing the return rate drastically. In our experiments for high-traffic products, the proposed method achieves 92.2% none-zero return rate compared to 86.5% from the top-performing existing method. This advantage is enlarged to 63.8% v.s. 3.9% for general products. As an approximate NNS method, our search method is able to achieve 90.7% recall rate against the exact 100-NNS compared to 74.2% or less recall from state-of-the-art packages. As for the computation efficiency, we achieve an average latency of less than 6ms for a product pool of 40 million products with a single machine of 61GB memory and a single Nvidia K80 GPU, and the method can scale up to support large product pools with multiple machines.

2 BACKGROUND AND OUR CONTRIBUTION

Product retrieval is fundamental for modeling and business applications in e-commerce. In order for relevance or ranking models to apply, various methods are adopted to retrieve relevant products, including the vastly used inverted-index based methods like Elasticsearch or Solr [12, 28]. In addition to supporting the downstream business applications, the retrieved products can also be consumed by instance-based product modeling [1, 35]. The retrieval method should be able to fetch enough products for a large proportion of the query products. The retrieval recall rate is also considered a good metric, but it is not commonly adopted because of the lack of ground truth [24]. In this paper, the proposed search method is largely related to two areas. Product representation learning and natural language processing help to convert a product to an embedding vector; the approximate nearest neighbor search supports similarity search based on product embeddings at scale.

Most of the existing product representation techniques for e-commerce depends largely on customer behavior data. Customer co-view and co-purchase information can be directly used to build product embedding such that products with similar browse or purchase history will share similar vectors [19]. Product catalog data, on the other hand, is a collection of free-form texts like title, description, brand, *etc.*. There are supervised and unsupervised ways to transform the catalog data into a vector space. The supervised way originates from ImageNet [8], where a multi-source neural network is trained to predict certain product categorization labels, and the last hidden layer is used as the instance embedding [5, 13, 27]. The categorization labels require extensive annotation while the embedding vector is found of high quality. The unsupervised way, on the other hand, applies to broader cases, especially when labels are unavailable. With the well established methods of word2vec and sentence2vec [7, 9, 16, 18], the problem of product representation learning effectively becomes the task of vector aggregation, namely the method of combining embedding vectors from multiple product fields. Though this problem seems fundamental for natural language processing in e-commerce, to the best of our knowledge, there is no related research dedicated to this problem.

Vector space search has drawn increasing attention because of its ubiquitous applications. In addition to product search, vector search is also frequently used in recommender systems [4, 17], and extreme classifications [30, 34]. Depending on the definition of the similarity metric, there are nearest neighbor search (NNS) on Euclidean distance, maximum cosine similarity search (MCSS), and maximum inner product search (MIPS) [26, 33]. While normalizing the embedding vectors will unify the three, there are independent works for each scenario [26]. The exact search method is not scalable since it requires calculating the pairwise similarity between the query and each of the candidates. Therefore, related works are focusing on approximate methods which can be roughly classified into tree-based approaches, hashing based approaches, and other approaches [3, 14, 15, 25, 26]. There are a large amount of works dedicated to high-performance approximate KNN, including FLANN, Iterative Expanding Hashing (IEH), Non-Metric Space Library (NMSLIB), and ANNOY [6, 11, 20, 21]. Our search method is a fingerprint-type method, belonging to the hashing based approach as IEH, while the other three methods are tree-based approaches.

We discuss technical details of the proposed search method in Section 3, followed by experiments regarding the return rate, the precision, and the recall according to exact KNN in Section 4. Section 5 discusses possible applications of the proposed method and concludes the paper. Before going into detailed discussions, we would like to highlight our contribution as follows:

- We have developed a product-to-product search method for reference product retrieval. The reference products are obtained via an optimized semantic hashing approach on product embedding generated by a novel attention auto-encoder neural network.
- With satisfactory precision, the proposed method is able to achieve considerably higher return rate compared to existing peers. Thus, our reference product search can support general downstream e-commerce applications.

- As an approximate KNN method, the proposed search algorithm is able to achieve high recall rate compared to state-of-the-art approximate KNN packages.
- It is flexible to adjust the number of returned candidate products based on the requirement of the applications. The latency-precision balance is adjustable in real-time to accommodate different use cases; the product embedding and retrieval encoding are also of a plug-and-play type.

3 METHOD

In this section, we first describe the offline and online processes of the proposed search method, followed by detailed discussions on product vectorizer and binary encoder that enable high-quality and fast search for products.

Two transformers are used to convert the text data of product catalog information into vector spaces: the product vectorizer $g(\cdot)$ and the binary encoder $h(\cdot)$. For a product p, the vectorizer converts the product catalog information into a d-dimension embedding (column-)vector $\vec{v} = g(p) \in \mathbb{R}^d$, and the binary encoder further transforms the embedding vector into a d'-dimension binary encoding vector $\vec{b} = h(\vec{v}) \in \mathbb{B}^{d'}$. We will introduce the design and training of the vectorizer $g(\cdot)$ and the encoder $h(\cdot)$ in Section 3.1 and 3.2, respectively. Let $\mathcal{P}$ denote the product pool where search results are retrieved from, and write $N := |\mathcal{P}|$ the size of $\mathcal{P}$. Vectorizing and the encoding processes are conducted offline for all the products in $\mathcal{P}$, resulting in an embedding set V and an encoding set B. For any binary encoding $\vec{b}_i \in B$, define the corresponding binary encoding bucket $\mathcal{B}_i := \{\vec{v} \in V \mid h(\vec{v}) = \vec{b}_i\}$. The set of all the buckets $\mathcal{B}^* := \{\mathcal{B}_1, \mathcal{B}_2, \dots\}$ defines a partition over V.

The online search process follows the semantic hashing mechanism [23]. We retrieve a small subset consisting of M ($M \ll N$) products from $\mathcal{P}$ via low-cost computation, followed by applying exact NNS to the candidate set to get the sorted search results. Such a schema approximates the exact NNS on the larger product pool $\mathcal{P}$ where M is adjusted to balance the quality of approximation and the latency. For any given query product p_q, our reference product search method proceeds as follows:

(1) Product vectorization. Calculate the query embedding $\vec{v}_q = g(p_q)$ and the encoding $\vec{b}_q = h(\vec{v}_q)$.
(2) Sort all binary buckets $\mathcal{B}_i$ in $\mathcal{B}^*$ according to the Hamming distance between its encoding $\vec{b}_i$ and the query encoding $\vec{b}_q$ to get a ranked list: $\mathcal{B}_{q_1}, \mathcal{B}_{q_2}, \mathcal{B}_{q_3}, \dots, \mathcal{B}_{q_{|\mathcal{B}^*|}}$.
(3) Find the minimal cut-off position c such that $|\mathcal{B}_{q_1}| + |\mathcal{B}_{q_2}| + \cdots + |\mathcal{B}_{q_c}| \geq M$.
(4) Conduct exact NNS for $\vec{v}_q$ and candidate set $\cup_{i=1}^{c} \mathcal{B}_{q_i}$ and return the top K ($K \leq M$) products.

Note that we conduct exact NNS for the retrieved M products and returns the top-K results as reference products. In reality, we usually apply a cut-off threshold γ to further filter out low quality products to ensure the precision of the search results. For a specific ranking or relevance model, the search results are further utilized or consumed. Intuitively, if the retrieved set covers a large proportion of the actual M-nearest neighbors of $\vec{v}_q$ in V, then the approximation should be good enough for downstream ranking models. We will discuss how to optimize the encoder $h(\cdot)$ for better search quality in Section 3.2.

Furthermore, we would like to highlight that the product pool can be expanded incrementally without retraining the vectorizer and the encoder. This is advantageous since the time-consuming offline pre-processing is not frequently conducted.

3.1 Product Representation Learning

The product vectorizer $g(\cdot)$ converts the product catalog data to a vector by incorporating signals from multiple attributes. A single product catalog field is essentially a sentence or a paragraph, and simple word2vec or sentence2vec model can be applied. Though there are supervised approaches to obtain product embeddings, we choose the unsupervised approach because it can be easily applied to billions of products without extensive human annotation efforts. We train a fastText [16] model on all the product information (around 400B tokens), and use this model as the field vectorizer. Each product catalog field is then converted into a vector $\vec{u} \in \mathbb{R}^d$.

The challenge is how to combine embedding vectors $\vec{u}^1, \vec{u}^2, \dots, \vec{u}^m$ from multiple fields. Taking more fields into consideration helps utilizing more signals from the product information, but naively concatenating field vectors suffers from two subsequent problems:

- The curse of dimensionality: more fields mean higher dimension of the product embedding, which increases latency.
- Missing fields and low-quality catalog information: we observe 10% to 45% missing rates for several important product fields, not to mention poorly-written fields with irrelevant or duplicated information.

Notice that the data quality issue does not come from data collection and processing. Therefore, it should be alleviated via proper modeling. To that end, we propose a novel attention auto-encoder network for embedding aggregation. The goal is that we smartly assign a weight $\alpha_j(U)$ to each field vector $\vec{u}^j \in \mathbb{R}^d$ so that the product embedding is a proper convex combination $\vec{v}_p = g(p) := \sum_{j=1}^{m} \alpha_j \vec{u}^j$, where the matrix $U := [\vec{u}^1, \vec{u}^2, \dots, \vec{u}^m]$. The averaging resolves the curse of dimensionality, and the embedding quality is optimized via minimizing information loss during the averaging. To be more specific, the weight vector $\vec{\alpha}(U)$ comes from a self-attention module defined as a softmax distribution:

$$\vec{\alpha}_j(U) := \exp(\phi(\vec{u}^j)) \Big/ \sum_{i=1}^{m} \exp(\phi(\vec{u}^i)), \tag{1}$$

where $\phi(\cdot)$ is a simple fully-connected neural network with one hidden layer and a scaler output: $\phi(\vec{u}^j) := \vec{\eta} \tanh(W\vec{u}^j + \vec{\xi})$. The attention module "looks at" each catalog field, returns a score estimating the relative amount of information from that field, and the softmax function further adjusts the weights for balance. The parameters $W, \vec{\xi}, \vec{\eta}$ are learned via minimizing the information loss from U to $\vec{v}$, or in other words, to ensure the combined vector $\vec{v}$ can best recover U from another two-layer neural network, defined as

$$\varphi(U) := \sigma\left(W^{(1)}\vec{v}_p + \vec{\xi}^{(1)}\right) = \sigma\left(W^{(1)}U\vec{\alpha}(U) + \vec{\xi}^{(1)}\right), \tag{2}$$

$$\vec{\hat{u}}^j(U) := W_j^{(2)}\varphi(U) + \vec{\xi}^{(2)}, \tag{3}$$

where the activation function $\sigma(\cdot)$ is the sigmoid function. The loss for the neural network is simply the mean squared error:

$$L := \frac{1}{m \times N} \sum_{p \in \mathcal{P}} \sum_{j=1}^{m} \left\| \vec{u}_p^j - \hat{u}_p^j(U) \right\|^2, \tag{4}$$

where the loss function L depends on matrices $W, W^{(1)}, W_j^{(2)} (1 \le j \le m)$ and vectors $\vec{\xi}, \vec{\xi}^{(1)}, \vec{\xi}^{(2)}, \vec{\eta}$. During training, U is generated from the catalog data of 40 million products, the dimension of the hidden layer is 32 for the attention module, 64 for $\varphi(U)$, and the model parameters are optimized via Adam Optimizer [24]. Note that for product embedding $g(\cdot)$, we only need the parameters $W, \vec{\xi}, \vec{\eta}$ in the attention module. The rest of the model parameters are auxiliary to help tune the attention module. For a given product with field embedding U, the attention module gives the optimal weights $\vec{\alpha}(U)$ so that the product embedding $\vec{v} = U\vec{\alpha}(U)$ can best incorporate and recover signals from U. The detailed neural network structure is demonstrated in Figure 1.

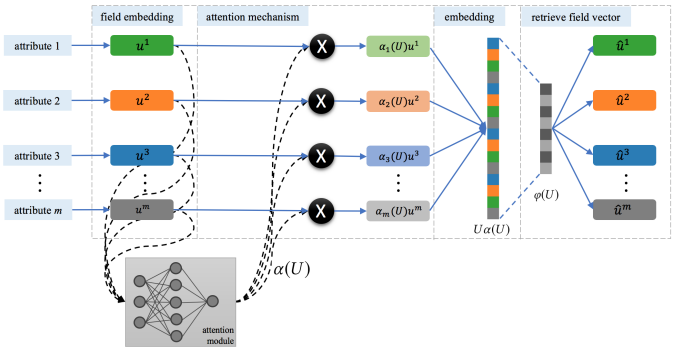

Figure 1: Model structure of the attention auto-encoder.

We conduct extensive experiments to evaluate the quality of embeddings based on the proposed attention auto-encoder (AAE) method. It is worth noting that the auto-encoder here is irrelevant to the auto-encoder trained for the binary encoding vector $\vec{b}$ in Section 3.2. The AAE method outperforms the state-of-the-art embedding methods. For example, we apply exact K-NNS to retrieve products from an annotated private dataset of 100k products in 1300 categories. The performance is measured by precision, which calculates the proportion of retrieved products from the same product category as the query product. The precision of the exact 3-NNS using the AAE embedding is 0.744, while the corresponding precisions for vanilla fastText, universal sentence encoder, and Bert are 0.662, 0.651, and 0.615, respectively [7, 9, 16]. We omit more details due to the page limit. Instead, we choose to conduct experiments directly on the final search method in Section 4.

3.2 Binary Hashing via Denoising Autoencoder

In this subsection, we demonstrate how to obtain the binary encoding vector $\vec{b} = h(\vec{v})$. The basic idea is to use denoising auto-encoder to compress the embedding $\vec{v}$ into a lower dimension logit vector $\vec{z}$ and find an optimal thresholding vector $\vec{\theta}$ to convert $\vec{z}$ into binaries. The binary encoding schema originates from Hinton *et al.* [23]. Our work focuses on optimizing the threshold $\vec{\theta}$ for better retrieval quality and lower latency. To enhance the search method precision and latency, the following desired properties of the encoding are proposed:

(1) The encoding dimension should be low for efficiency.
(2) Tiny buckets should be avoided, otherwise NNS has to take place on too many buckets and latency issue will come up.
(3) Huge buckets should be avoided for ranking efficiency.
(4) The average bucket size should be controllable for different values of M.

Denoising auto-encoders are frequently used for extracting and composing robust features, where the neural network is fed with manually corrupted data to enhance feature quality and stability [31, 32]. The details of the network structure is well-known to the community, and hence we only demonstrate how to optimize the thresholding vector $\vec{\theta}$ in this paper. Our auto-encoder model consists of a two-layer encoder and a two-layer decoder, where the number of the encoded feature layer d' is set to 32 with sigmoid function as the activation. We use $\vec{z} = [z_1, z_2, \ldots, z_{32}]$ to denote the compressed feature from the encoding layer where $z_i \in (0, 1)$ $\forall 1 \le i \le 32$, and the final binary encoding is obtained via thresholding by $b_i = \mathbb{1}_{z_i \ge \theta_i}$.

Naively choosing $\theta_i = 0.5$ as in [23] will result in too many buckets with extreme sizes. Note that now the binary bucket $\mathcal{B} = \mathcal{B}(\vec{\theta})$ depends on the thresholding vector, and let $n_i(\vec{\theta}) = |\mathcal{B}_i(\vec{\theta})|$ be the bucket size. We define the following objective function:

$$L'(\vec{\theta}) := \sum_{i}^{|\mathcal{B}^*|} \left(\frac{\chi^4}{n_i^2(\vec{\theta})} + n_i^2(\vec{\theta}) \right), \tag{5}$$

where χ is a tunable parameter to balance the average bucket size and the number of buckets. For a single bucket, the summand in (5) is convex and has a unique minimum at $n_i = \chi$, and both huge and tiny n_i are naturally penalized. However, the objective (5) becomes non-convex and even discontinuous with respect to $\vec{\theta}$, making traditional optimization techniques unsuitable. To overcome the optimization difficulty, we adopt a continuous genetic algorithm [2], with population size 100, mutation rate 0.2, and in each generation 100 pairs are randomly selected for crossover. We iterate for 200 generations and choose the $\vec{\theta}$ with the smallest objective L' across all the generations. For our test product pool of 40 million products, the optimized $\vec{\theta}$ for $\chi = 100$ is able to increase the average bucket size from 1.78 via the vanilla semantic hashing to 57.32, and the largest bucket size decreases from 60k to 3.5k. Again, we omit the standalone evaluation of the proposed encoding method, and conduct experiments on the overall search method in Section 4.

4 EXPERIMENTS

In this section, we present a series of experiments to evaluate the proposed solution. We have built a Reference Product Service (RPS) using the AAE product embedding and semantic hashing approximate NNS. For a given query product, its embedding and binary encoding are computed in real-time, followed by the semantic hashing NNS, which is implemented using a CPU-GPU hybrid model. We first compare RPS with its peer services and present two quality metrics, namely the return rate and the precision at K, in Section

Table 1: Return rate at K, $\mathcal{R}_K(\mathcal{S}, Q_{\text{purchased}})$

K	RPS	Method-1	Method-2	Behavior-1	Behavior-2
1	0.922	0.293	0.865	0.643	0.742
3	0.902	0.293	0.748	0.528	0.688
5	0.893	0.149	0.637	0.475	0.657
10	0.888	0.0	0.408	0.405	0.606
50	0.775	0.0	0.0	0.234	0.415

Table 2: Return rate at K, $\mathcal{R}_K(\mathcal{S}, Q_{\text{general}})$

K	RPS	Method-1	Method-2	Behavior-1	Behavior-2
1	0.638	0.008	0.039	0.017	0.030
3	0.599	0.008	0.030	0.013	0.023
5	0.586	0.003	0.024	0.011	0.020
10	0.579	0.00	0.014	0.009	0.017
50	0.456	0.00	0.00	0.004	0.010

4.1 and 4.2, respectively. Then, as the semantic hashing NNS approximates the Exact NNS, we compare its performance with the Exact NNS in Section 4.3. Lastly, the complexity and computation efficiency are briefly discussed in Section 4.4.

4.1 Return Rate Test

For a given test pool of queries Q_{test}, let $\psi(\mathcal{S}, q)$ be the number of search results returned by $\mathcal{S}$ for the query product $q \in Q_{\text{test}}$. Then the return rate at K, denoted by $\mathcal{R}_K(\mathcal{S}, Q_{\text{test}})$ is defined as

$$\mathcal{R}_K(\mathcal{S}, Q_{\text{test}}) := \frac{1}{|Q_{\text{test}}|} \sum_{q \in Q_{\text{test}}} \mathbb{1}_{\psi(\mathcal{S}, q) \geq K}, \tag{6}$$

where $\mathbb{1}$ is the indicator function. The return rate estimates the ability to retrieve enough reference products for different query products. In reality, this ability is largely limited by product feature availability and the design of the search method. For example, a search method depending on customer browsing history will not work for a new product without views. However, recall that one of our goals of the reference products is to make it general so it is able to support different applications.

Two test sets are constructed, namely the purchased set and the general set. The purchased set consists of 10^6 randomly sampled products with purchase history. The general set consists of 10^6 products randomly sampled from a billion-level product pool. Products from the purchased set can be considered be higher quality. We compare the return rate of RPS with four existing product-to-product services. Two peer services highly depend on behavioral data (denoted by Behavior-1 and Behavior-2), and the other two depend on general product information (denoted by Method-1 and Method-2). RPS uses a product pool of 4×10^7 products with $d = 100$, $d' = 32$ and $M = 4000$. The related results are listed in Table 1 for the purchased query set and Table 2 for the general query set.

We acknowledge that we have no control on the quality or the size of the product pool used by each peer service, nor how the search method is implemented. We simply summarize the observed results as below. Our service returns enough search results for a larger proportion of query products. This advantage becomes more obvious for general products where all the peer services fail. Recall that the return rate estimates the applicability of a solution, better return rates indicate that our solution is able to support broader applications.

Table 3: Precision test in each product line

Product Line	Method-1	Method-2	RPS
Softline	0.582	0.572	0.865
Hardline	0.621	0.656	0.794
Consumable	0.673	0.705	0.827

4.2 Precision Test

The precision at K metric $\mathcal{P}_K$ is defined as the proportion of the top-K results that are indeed similar to the query based on human judgement. More specifically, let $\varphi(\mathcal{S}, p, K)$ be the number of positively annotated products from the top-K search results by method $\mathcal{S}$ for a query $q \in Q_{\text{test}}$, the precision at K metric $\mathcal{P}_K$ is defined as

$$\mathcal{P}_K(\mathcal{S}, Q_{\text{test}}) := \frac{1}{|Q_{\text{test}}|} \sum_{q \in Q_{\text{test}}} \varphi(\mathcal{F}, q, K). \tag{7}$$

We observe a sizable proportion of the search results from Behavior-1 and Behavior-2 is dominated by noise since both methods heavily depend on behavioral features, making them less competitive for the precision test. Thus, we only annotate the search results from Method-1, Method-2, and our own RPS. We set $K = 5$ and randomly sample 1500 products such that $\psi(\mathcal{S}, q) \geq 5$ for all the three methods. In order to better demonstrate the difference between the methods, the annotators are asked to use strict criterion in terms of similarity and to label at least 3 products as negative out of the 15 products pooled from the top-5 results from each of the three method. The precision test results are shown in Table 3 where RPS outperforms the other two baseline peers. Note that the results should be viewed for comparison purpose only and the actual precision is higher for all the methods because of the strict criterion.

4.3 Approximate KNN Recall Test

In addition to the above search quality test, we also conduct the recall test for completeness. The recall rate measures the proportion of actual positives that are included in the search results, which is generally not practical since the ground truth is unknown. Instead, we follow the convention to compute the recall against the exact NNS method and compare it with four other open-source approximate NNS methods. Non-Metric Space Library (NMSLIB) [22], Approximate Nearest Neighbors Oh Yeah (ANNOY) [10], and Fast Library for Approximate Nearest Neighbors (FLANN) are tree-based approaches, while Iterative Expanding Hashing is a hashing type approach [6, 11, 20, 21].

Since we require the exact KNN results to be the ground truth, we use a smaller dataset for the recall test: a set of 2 million products generated randomly. For each method, we calculate the proportion of the top-K search results that actually hit the top-K products from exact KNN search. Recall values are presented in Table 4 for K

Table 4: Recall-at-K test

K	RPS	NMSLIB	FLANN	ANNOY	IEH
1	97.24	90.13	90.59	91.35	95.14
5	95.82	93.42	85.57	85.94	90.88
10	94.85	92.67	79.10	82.62	87.65
100	90.72	61.90	58.64	66.90	74.21

varying from 1 to 100. Note that the query itself is always excluded from the search results.

Table 4 shows that our approximate KNN based on binary encoding achieves satisfactory recall rates. In addition, we would like to highlight its flexibility. The efficiency-quality parameter M can be adjusted online without preprocessing to balance efficiency and quality, while most of other approximate KNN algorithms can not. It also allows easy modification of product pool in realtime. For example, adding new embedding and encoding vectors are more easily handled than training a new tree structure.

4.4 Latency

In this subsection, we briefly discuss the efficiency and the latency of our search method. In general, the combined embedding and encoding latency for a single query is under 1ms. With a single AWS machine of type p2.xlarge (equipped with an NVIDIA K80 GPU), for a product pool of 40 million products and the subset size $M = 4000$, the average search latency is under 6ms, which is in the same scale as the general network latency. Thus, the proposed method is able to efficiently support general business applications.

5 DISCUSSION AND CONCLUSION

In this paper, we propose a search method to surface a set of reference products. The product catalog information is transformed into a high-quality embedding of low dimension via a novel attention auto-encoder neural network, and the embedding is further coupled with a binary encoding vector for high quality vector search at scale. We conduct extensive experiments to evaluation the return rate, the precision, and the recall rate of the proposed method, and compare them with peer methods. Since our method is able to yield a satisfactory number of high-quality results for most of the query products, the reference products are readily consumable for various business ranking models to support applications like pricing, substitution, and recommendation. We believe such an acceleration to support multiple applications will bring fundamental values to the e-commerce business, and invite more future works on the algorithms and applications of the reference product set.

REFERENCES

[1] David W Aha, Dennis Kibler, and Marc K Albert. 1991. Instance-based learning algorithms. *Machine learning* 6, 1 (1991), 37–66.

[2] Christine M Anderson-Cook. 2005. Practical genetic algorithms.

[3] Alex Auvolat, Sarath Chandar, Pascal Vincent, Hugo Larochelle, and Yoshua Bengio. 2015. Clustering is efficient for approximate maximum inner product search. *arXiv:1507.05910* (2015).

[4] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender systems*. ACM, 257–264.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[6] Leonid Boytsov and Bilegsaikhan Naidan. 2013. Engineering efficient and effective non-metric space library. In *International Conference on Similarity Search and Applications*. Springer, 280–293.

[7] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv:1803.11175* (2018).

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee, 248–255.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805* (2018).

[10] Bernhardsson Erik. 2016. Approximate Nearest Neigbors OnYeah (Annoy). https://github.com/spotify/annoy.

[11] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2013. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 12 (2013), 2916–2929.

[12] Clinton Gormley and Zachary Tong. 2015. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine.* " O'Reilly Media, Inc.".

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[14] Gisli R Hjaltason and Hanan Samet. 2003. Index-driven similarity search in metric spaces (survey article). *ACM Transactions on Database Systems (TODS)* 28, 4 (2003), 517–580.

[15] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2011), 117–128.

[16] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv:1607.01759* (2016).

[17] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *International conference on Information and knowledge management*. ACM, 535–544.

[18] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International Conference on Machine Learning*. 1188–1196.

[19] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 1 (2003), 76–80.

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

[21] Marius Muja and David G Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 11 (2014), 2227–2240.

[22] Bilegsaikhan Naidan, Leonid Boytsov, Malkov Yury, Novak David, and Frederickson Ben. 2016. Non-Metric Space Library (NMSLIB). https://github.com/nmslib/nmslib.

[23] Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Semantic hashing. *International Journal of Approximate Reasoning* 50, 7 (2009), 969–978.

[24] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*. Vol. 39. Cambridge University Press.

[25] Thomas Seidl and Hans-Peter Kriegel. 1998. Optimal multi-step k-nearest neighbor search. In *ACM Sigmod Record*, Vol. 27. ACM, 154–165.

[26] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Advances in Neural Information Processing Systems*. 2321–2329.

[27] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556* (2014).

[28] David Smiley, Eric Pugh, Kranti Parisa, and Matt Mitchell. 2015. *Apache Solr enterprise search server*. Packt Publishing Ltd.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[30] Sudheendra Vijayanarasimhan, Jonathon Shlens, Rajat Monga, and Jay Yagnik. 2014. Deep networks with large output spaces. *arXiv:1412.7479* (2014).

[31] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*. ACM, 1096–1103.

[32] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research* 11, Dec (2010), 3371–3408.

[33] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. 2014. Hashing for similarity search: A survey. *arXiv:1408.2927* (2014).
[34] Jason Weston, Samy Bengio, and Nicolas Usunier. [n. d.]. Wsabie: Scaling up to large vocabulary image annotation.
[35] D Randall Wilson and Tony R Martinez. 2000. Reduction techniques for instance-based learning algorithms. *Machine learning* 38, 3 (2000), 257–286.
[36] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. 2014. Supervised hashing for image retrieval via image representation learning.