Text Matching Indexers in Taobao Search

Sen Li*
Alibaba Group
Hangzhou, China
lisen.lisen@taobao.com

Fuyu Lv
Alibaba Group
Hangzhou, China
fuyu.lfy@taobao.com

Ruqing Zhang
CAS Key Lab of Network Data
Science and Technology, ICT, CAS
Beijing, China
zhangruqing@ict.ac.cn

Dan Ou
Alibaba Group
Hangzhou, China
oudan.od@taobao.com

Zhixuan Zhang
Alibaba Group
Hangzhou, China
zhibing.zzx@taobao.com

Maarten de Rijke
University of Amsterdam
Amsterdam, The Netherlands
m.derijke@uva.nl

ABSTRACT

Product search is an important service on Taobao, the largest e-commerce platform in China. Through this service, users can easily find products relevant to their specific needs. Coping with billion-size query loads, Taobao product search has traditionally relied on classical term-based retrieval models due to their powerful and interpretable indexes. In essence, efficient retrieval hinges on the proper storage of the inverted index. Recent successes involve reducing the size (pruning) of the inverted index but the construction and deployment of lossless static index pruning in practical product search still pose non-trivial challenges.

In this work, we introduce a novel **SM**art **IND**exing (`SMIND`) solution in Taobao product search. `SMIND` is designed to reduce information loss during the static pruning process by incorporating user search preferences. Specifically, we first construct "user-query-item" hypergraphs for four different search preferences, namely purchase, click, exposure, and relevance. Then, we develop an efficient TermRank algorithm applied to these hypergraphs, to preserve relevant items based on specific user preferences during the pruning of the inverted indexer. Our approach offers fresh insights into the field of product search, emphasizing that term dependencies in user search preferences go beyond mere text relevance. Moreover, to address the vocabulary mismatch problem inherent in term-based models, we also incorporate an multi-granularity semantic retrieval model to facilitate semantic matching. Empirical results from both offline evaluation and online A/B tests showcase the superiority of `SMIND` over state-of-the-art methods, especially in commerce metrics with significant improvements of 1.34% in Pay Order Count and 1.50% in Gross Merchandise Value. Besides, `SMIND` effectively mitigates the Matthew effect of user queries and has been in service for hundreds of millions of daily users since November 2022.

*Corresponding author.

KDD '24, August 25–29, 2024, Barcelona, Spain
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671654

CCS CONCEPTS

• **Information systems → Top-k retrieval in databases**.

KEYWORDS

Product search, Static pruning inverted index, Semantic matching

ACM Reference Format:
Sen Li, Fuyu Lv, Ruqing Zhang, Dan Ou, Zhixuan Zhang, and Maarten de Rijke. 2024. Text Matching Indexers in Taobao Search. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3671654

1 INTRODUCTION

E-commerce platforms like Amazon and Taobao have become part of people's daily lives. In Taobao, the product search engine serves hundreds of millions of users, by providing the top-k personalized relevant items from billion-scale candidate products, within hundreds of milliseconds. The classic industry product search architecture, known as "index-retrieval-prerank-rank-rerank," includes three sequential steps: (1) indexing: to build an index for each item in the corpus; (2) retrieval: to retrieve an initial set of candidate items for a user query; and (3) prerank-rank-rerank: to calculate a personalized relevance score of each candidate. Of these, the indexing and retrieval stages are pivotal as they determine the upper bounds of the search performance.

To handle billions of queries and products, Taobao predominantly employs classical term-based matching techniques, emphasizing *interpretability* and *high search relevance.* Specifically, in indexing, we construct traditional inverted indexes for the text of each item. During retrieval, we integrate the rewritten query [22] with the original user query to mitigate vocabulary mismatches [32, 39, 51]. Subsequently, we perform exact matching between the query and the items. To return all matched candidates, a popular method is to employ a dynamic index pruning scheme [3, 14]. However, in practical scenarios, this approach can prove to be time, storage, and computation-intensive. This is especially significant in Taobao product search, where a substantial portion of latency is dedicated to the "prerank-rank-rerank" phase for personalization modeling. Additionally, users tend to focus on a limited number of top-k items for a query and often issue a new query if the initial results fail to meet their expectations.

Hence, in Taobao product search, a lightweight static pruning inverted index scheme [3, 4, 38], a.k.a. static term-based pruning, is widely employed, saving **50%** of system resource. This scheme comprises two parts: the pruned inverted index, denoted as I_P, and the full inverted index, denoted as I_F. I_P consists of the posting lists (an occurrence of a term within the text an item is called a posting) for each term but truncated with a given length, while I_F is the regular full posting lists. A common practice for pruning a posting list is to consider term dependencies related to textual relevance. In web search, Carmel et al. [4] proposed using the TF-IDF scoring function $f(t,i)$ to preserve the textually most relevant items for reducing the truncation loss of each term. DeepCT [8], SPARTA [50], and SPLADE [15, 16], are recently proposed methods to provide contextualized term dependencies of text relevance signal.

However, pruning based on the general textual relevance term dependencies is not sufficient for product search. To elaborate, many merchants employ search engine optimization (a.k.a. SEO) to increase their visibility by using relevant and popular keywords to describe their products . This practice, in turn, diminishes the efficacy of textual relevance. Additionally, existing work overlooks the term dependencies of User Search Preferences (USP), as discussed in Section 4.5.4 and 4.5.5. This oversight results in the incapacity to explicitly retain user-preferred items within the truncated postings lists. Through an analysis of user logs, we found that the majority of users made purchases based on specific query terms. To illustrate, consider an item titled "蕾丝薄款连衣裙 (Self-portrait Lace Thin Dress)." Users did not click or purchase it with the textual relevant term "薄款 (thin)," but with combinations of the remaining terms. This phenomenon highlights the existence of higher-order term-item dependencies within USP, transcending mere text relevance. These higher-order term dependencies not only assist in mitigating SEO practices by excluding certain textually relevant terms often added by merchants, but also have the potential to minimize information loss by retaining user-preferred items. However, how to explicitly capture and incorporate these dependencies into the static pruning process while mitigating the noise inherent in user logs, remains relatively unexplored.

Therefore, in this paper, we propose an efficient billion-scale product search indexing solution, named SMart INDexing (SMIND). The key idea is to minimize information loss during the static inverted index pruning process, by incorporating higher-order term dependencies from user search preferences. Concretely, we first investigate the user search process and define four basic preference patterns, namely purchase, click, exposure, and relevance preference pattern. Subsequently, we utilize extensive user logs and employ ternary "user-query-item (u,q,i)" interactions to build four hypergraphs [7], i.e., user search purchase, click, exposure, and relevance preferences hypergraphs. We then decompose each hypergraph (u,q,i) into a term-level one (u,q_t,i_t) by representing the vertex of query q and item i with their text terms. Then, we devise an efficient TermRank algorithm to explicitly capture the higher-order term-item dependencies within USP. Finally, during the static pruning process, to alleviate the inherent noise in user behavior, we formulate a user-aware term scoring function $f(t,i)$ by considering both the term dependencies within USP and an additional query-independent item quality factor, to rank items within the postings lists and preserve the top items.

Given the diverse nature of user demands, it is intractable for query rewriting to address the vocabulary mismatch problem on its own. For instance, when a user enters a query like "衣服 (clothes)," they may actually intend to purchase a "连衣裙 (dress)." Additionally, the inconsistent contextual alignment between the query and the item can lead to variations in the Chinese tokenization of the same text. This may adversely affect the performance of both the original user queries and their rewritten counterparts. To this end, we further incorporate a multi-grained semantic neural retrieval model to facilitate fine-grained contextualized interactions between query and item representations. It is combined with the original user queries and their rewritten versions to enhance semantic matching capabilities, thereby improving recall for potentially missed items.

To the best of our knowledge, SMIND is the first solution to explore the use of term dependencies in user search preferences for pruning inverted indexes, and effectively combines query rewriting with an exact and fuzzy match indexer. The effectiveness of SMIND is verified by extensive offline qualitative and quantitative analyses, and large-scale online A/B tests. We observed that for torso and tail queries, SMIND enables text matching indexers to deliver high-quality items aligned with user search intentions. This prevents users from shifting their queries to top queries, thereby alleviating the Matthew effect [36] of user queries.

The main contributions of this work are as follows:

(1) We introduce SMIND, an efficient and effective billion-scale product search indexing solution that includes a user search preferences-aware pruning inverted indexer and a semantic indexer, to enable the retrieval of relevant, high-quality items.
(2) We conduct a term-level analysis of multiple user search preferences, represented as (u,q_t,i_t) hypergraphs, which are employed to prune inverted indexes and bridge the gap between user queries and pruned inverted indexes.
(3) We offer new insights that user search preferences encompass higher-order term dependencies beyond text relevance.
(4) We evaluate SMIND's performance through comprehensive offline qualitative and quantitative analyses, as well as large-scale online A/B tests conducted on Taobao Product Search.

2 RELATED WORK

2.1 Inverted Indexer

In a typical search engine efficient text/product retrieval is achieved through the use of an inverted index structure [51], which associates terms with relevant documents in the collection. For each term in the document collection, a posting list of documents is stored. The index file grows with increases in the size of the document collection, which is problematic for fast and inexpensive query evaluation in an online system [26]. Thus, effective pruning techniques are needed to reduce the size of inverted indexes. Such techniques generally fall into two categories, i.e., dynamic and static. Dynamic pruning [3, 11, 14, 41] decides, during query evaluation, whether certain terms or document postings should be included in the ranked list, until some stopping condition is met. Static techniques [4, 26, 34, 38] remove terms or items from the full index in advance to take advantage of a two-tiered index. The small, yet effective index that is left is called the pruned index. Static and dynamic pruning techniques can complement each other, since searching the pruned index can be accelerated by dynamic pruning for a specific query. Panigrahi and Gollapudi [35] exploit the structural characteristics of commerce search queries (i.e., from users searching for products on the web) for the selection of documents, which relies on manual labelling data to structure queries.

Our work focuses on static term-based index pruning at the posting-level. It removes items from each term whose contribution is so small that their removal has a minor impact on the retrieval system. Carmel et al. [4] examine an idealized term-based pruning algorithm. The score of a term for a document is formulated by term dependencies of text relevance, i.e., TF-IDF [37] scoring function. The pruned index has been proved, to some extent, to provide results that are almost as good as search results derived from the full index. Recently, learned sparse representation methods such as DeepCT [8], SPARTA [50], and SPLADE [15, 16], are being used to provide contextualized term dependencies of text relevance signal. However, they have been shown to frequently assign suboptimal weights to terms [28, 29, 31], which may degrade search relevance as we show below.

Long and Suel [26] propose to integrate global linked-based ranking (e.g., PageRank [2]) and term-based text relevance ranking as the final term-doc scoring function. Two optimized pruning techniques, namely keyword-based and document-based pruning, are studied by Ntoulas and Cho [34] to avoid any degradation of result quality. Rossi et al. [38] combine the dynamic pruning technique BMW [11] and a static two-tiered index (the pruned and full index) to accelerate query evaluation.

To the best of our knowledge, existing studies into index pruning are conducted primarily in the context of web search, where text relevance and hyperlink analysis are key to the final term-doc scoring function for pruning the postings lists. However, this is far from sufficient in a product search setting. How to exploit higher-order term dependencies of User Search Preferences (USP) for pruning while mitigating the noise inherent in user logs has not been addressed. Apart from using the term dependencies of USP from user click logs to reduce the size of inverted index, there are many graph-based dense retrieval works of modeling user click logs [7, 13, 33]. However, they have difficulties in learning exact matching signals, and thus tend to return irrelevant results when users search for exact queries [6, 23].

2.2 Semantic Indexer

There is a rich literature [19, 40, 48] on capturing the semantics of queries and documents for the first-stage retrieval. For an overview, we refer to [17]. Building on this work, several large-scale deployments of semantic search systems have been reported. Nigam et al. [32, Amazon] develop a two-tower model to address the vocabulary mismatch of a lexical matching engine. One side uses n-gram query features and the other side exploits item features. Chang et al. [5, Amazon] use a tree-based extreme multi-label classification model to improve inference latency. Huang et al. [18, Meta] describe their embedding-based retrieval system integrated with term matching-based retrieval, which considers query texts and searchers' context factors together. Liu et al. [25, Meta] and Yu et al. [47, Meta] apply multi-modal learning and pre-training techniques to their social product search. Also using pre-training, Liu et al. [24, Baidu] develop a retrieval model based on ERNIE [42], a state-of-the-art Chinese pre-training model. Recently, Magnani et al. [30, Walmart] have presented a hybrid retrieval system for e-commerce retail search. They combine the full inverted index and an embedding-based neural retrieval to better serve the traffic of tail queries. However, the search setting of Walmart is relatively small, withe million-scale query loads and a small collection of items. With billion-scale products, the search scenario requires careful trade-offs between efficiency and various cost factors. Below, we discuss the billion-scale product search indexing solution in Taobao Product Search.

3 PROPOSED METHOD

Here, we will detail the user search preferences-aware pruned inverted indexer and semantic indexer of SMIND shown in Figure 1, respectively. The retrieval process is illustrated in Section B.1 of the Appendix. For notation, let $U = \{u_1, \ldots, u_N\}$ be a set of N users, $Q = \{q_1, \ldots, q_L\}$ the user queries, $I = \{i_1, \ldots, i_M\}$ a collection of items, and $T = \{t_1, \ldots, t_K\}$ the unique terms of collection I. See Section A of the Appendix for the problem definition.

3.1 User Search Preferences-Aware Pruned Inverted Indexer

In this section, we study how to improve the static term-based pruning by using higher-order term dependencies of user search preference patterns USP and query-independent item quality factor Q. First, we construct four term-level ternary "user-query-item (u, q, i)" interactions hypergraphs based on four basic patterns. Then, we develop an efficient TermRank algorithm to capture multiple term dependencies of USP patterns. Finally, we integrate the term dependencies of USP and Q into the final term-based scoring function for pruning the posting lists of each term.

3.1.1 HyperGraph Construction. A hypergraph is a more general topological structure than a bipartite graph, where an edge connects two or more nodes, named hyperedge. Given some ternary (u, q, i) interactions, it is intractable to construct a simple graph to preserve ternary relations [7], and we have a hypergraph $G = (V, E)$ including: (1) a node set V whose elements represent a user, query, item (i.e., $V = U \cup Q \cup I$); and (2) a hyperedge set E whose elements represent a ternary relation depicting the existence of an interaction among them. We construct multiple hypergraphs G from user logs based on the defined patterns f among ternary (u, q, i), i.e.,

$$G = f(u, q, i), \tag{1}$$

where f establishes the hyperedge among (u, q, i) according to a certain pattern, as follows:

$G_{pur}/G_{clk}/G_{expo}$ for the user purchase/click/exposure preference pattern $f_{pur}/f_{clk}/f_{expo}$, each hyperedge represents how a user initiates a query and then purchases/clicks/sees an item.

G_{rel} for the user relevance pattern f_{rel}, each hyperedge represents a relevant relation among (u, q, i), where i is an item, in the rank phase of search system, is associated with a user and query.

Term-Level Decomposition. Because G describe USP patterns at the node-level (u, q, i) rather than the term-level, we further decompose each node into the term-level by replacing V's query and item elements with their terms q_t and i_t, i.e., for a query q_2 (Figure 2(b))

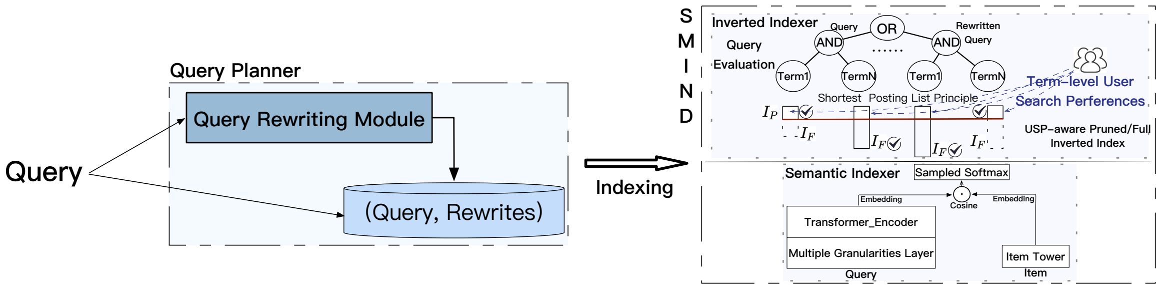


Figure 1: General structure of the proposed large-scale Smart Indexing (SMIND) hybrid indexer solution and its combinations of query rewriting technique.

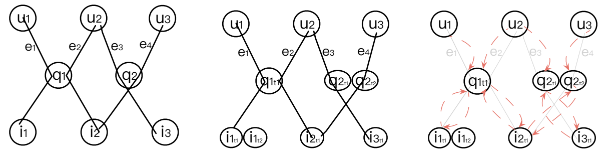


Figure 2: Given four interactions (u_1, q_1, i_1), (u_2, q_1, i_2), (u_2, q_2, i_3) and (u_3, q_2, i_2). (a) A sample of hypergraph whose hyperedge $e_i (i = 1, 2, 3, 4)$ is a user-query-item interaction; (b) Assuming q_1=$\{t1\}$, $q2$=$\{t1, t2\}$, i_1=$\{t1, t2\}$, i_2=$\{t1\}$, i_3=$\{t1\}$, we decompose (a) into term-level; (c) Assuming $q_{1_{t1}}$=$i_{1_{t1}}$=$i_{2_{t1}}$=$q_{2_{t2}}$, $q_{2_{t1}}$=$i_{3_{t1}}$, we create value flow based on each term-level hyperedge (u, q_t, i_t) .

with two terms $\{t_1, t_2\}$, then the element is $q_{2_{t_1}}$, $q_{2_{t_2}}$ instead of the whole query q_2. We do the same on the item side.

Now, the hyperedge is able to describe the exact term match between the user queries and item titles. For each ternary (u, q, i) interaction, we iterate over the combinations of terms in query and item and then establish a hyperedge (u, q_t, i_t) if and only if their terms are the same. Considering an interaction of (u_1, q_1 =$\{t_1\}$, i_1=$\{t_1, t_2\}$), there will be two candidates (i.e., (u_1, $q_{1_{t1}}$, $i_{1_{t1}}$) and (u_1, $q_{1_{t1}}$, $i_{1_{t2}}$)). Assuming $q_{1_{t1}}$=$i_{1_{t1}}$, we then establish a term-level hyperedge (u_1, $q_{1_{t1}}$, $i_{1_{t1}}$) as in Figure 2(b).

So far, we have constructed a term-level hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ including: (1) a node set $\mathcal{V}$ whose element represents u, q_t, i_t; and (2) a hyperedge set $\mathcal{E}$ whose element represents a ternary relation of u-q_t-i_t depicting how a user interacts with item via the term t. Obviously, $\mathcal{G}$ describes what terms users prefer to use to interact with an item, which is exactly the higher-order term-item dependencies of USP. We apply the above term-level decomposition process to four user search hypergraphs G_{pur}, G_{clk}, G_{expo}, and G_{rel}, obtaining term-level hypergraphs $\mathcal{G}_{pur}$, $\mathcal{G}_{clk}$, $\mathcal{G}_{expo}$, and $\mathcal{G}_{rel}$.

3.1.2 TermRank.

3.1.2 TermRank. We introduce an efficient TermRank algorithm to capture the term-item dependencies of USP from every $\mathcal{G}$. First, we transform the undirected graph $\mathcal{G}$ into a directed one. As shown in Figure 2(c), for each hyperedge (u, q_t, i_t) of $\mathcal{G}$, we create a bidirectional connection between q_t and i_t nodes (*i.e.,* $q_t \leftrightarrow i_t$), while for u and q_t nodes we only link a one-way edge (*i.e.,* $u \rightarrow q_t$). Assuming there is a value for each node, we then generate a directed graph where values flow between nodes following hyperedge e. The idea is to obtain node i_t's value because it describes how much users prefer to use the term t to interact with item i, i.e., term-item dependencies of USP. The values of i_t and q_t are calculated by:

$$v(i_t) = \frac{\sum_{e \in \mathcal{E}_{i_t}} v(q_t) \cdot v(u)}{\sum_{t \in i} v(i_t)}, \tag{2}$$

$$v(q_t) = \frac{\sum_{e \in \mathcal{E}_{q_t}} v(i_t) \cdot \alpha_i}{\sum_{t \in q} v(q_t)}, \tag{3}$$

where $v(\cdot)$ is the value function. $\mathcal{E}_{i_t}$ and $\mathcal{E}_{q_t}$ denote the set of hyperedges containing i_t and q_t, respectively. Empirically, $v(u)$ and α_i are the normalized values of the user$_u$'s search times and item$_i$ exposure times. The denominator in Eqn. (2) and (3) normalizes the values of all terms within an item/query for numerical stability.

Eqn. (2) and (3) are an alternate iterative process, which is denoted as the TermRank algorithm. However, since $\mathcal{G}$ is a non-strongly connected graph, when TermRank converges, the value will accumulate to some hot attribute terms (e.g., brand/category terms). This is not helpful to prune the postings lists. Therefore, we set the number of iterations as a hyper-parameter. Finally, we apply TermRank on four term-level hypergraphs $\mathcal{G}_{pur}$, $\mathcal{G}_{clk}$, $\mathcal{G}_{expo}$, $\mathcal{G}_{rel}$, obtaining term dependencies of user search purchase/click-/exposure/relevance preferences of each node i_t, i.e., $\mathcal{G}_{pur}\_v(i_t)$, $\mathcal{G}_{clk}\_v(i_t)$, $\mathcal{G}_{expo}\_v(i_t)$, $\mathcal{G}_{rel}\_v(i_t)$.

3.1.3 Integrating Term Dependencies of USP and Item Quality Factor Q.

The term dependencies of USP show which terms users prefer to use to interact with items. There is some amount of noise inherent in user logs [22, 23], making the user search preference-aware truncated posting lists less effective. To this end, we further include an item quality factor Q into the final $f(t, i)$ score function. Although PageRank is popularly used to measure the quality of web sources, it is not applied in e-commerce setting for items are considered to be independent objects. We use the historical trading volume of each item as a quality factor Q. Intuitively, the more transactions, the better the item quality. Given a term t, we construct a user-aware scoring function $f(t, i)$, by mixing the term dependencies of USP and a query-independent item quality factor Q, to truncate its full posting lists I_{t_f} to a given length. Mathematically,

$$t\_i_{USP} = \mathcal{G}_{pur}\_v(i_t) + \mathcal{G}_{clk}\_v(i_t) + \mathcal{G}_{expo}\_v(i_t) + \mathcal{G}_{rel}\_v(i_t) \tag{4}$$

$$f(t, i) = t\_i_{USP} \cdot i_Q, \tag{5}$$

where t_i_{USP} and i_Q denote the mixed USP score between the term t and the item i, and the item i's quality, respectively. For convenience, we use IVI-USP-Q to refer the pruned **I**n**V**erted **I**ndex based on USP and Q.

3.2 Semantic Indexer

In this section, we introduce the semantic dense indexer to reduce the mismatch between user queries and item. We will present query tower, item tower and loss function, respectively.

3.2.1 Query Tower. Queries in Taobao product search are mainly Chinese, which poses two challenge: (1) Because the context of query and item are different, the Chinese tokenization of the same text could be different; and (2) user queries are quite short. We train a two-tower semantic indexer and represent the queries in multiple semantic granularities to enrich its representations.

Multiple Granularities Layer. Specifically, for a query "红色连衣裙," we represent it as uni-gram $\{c_1, \ldots, c_m\}$ ({红, 色, 连, 衣, 裙}), bi-gram $\{c_1c_2, \ldots, c_{m-1}c_m\}$ ({红色, 色连, 连衣, 衣裙}), and semantic word segmentation $\{t_1, \ldots, t_n\}$ ({红色, 连衣裙}) granularities. We thus obtain four granular representations $h_q \in \mathbb{R}^{4\times d}$ by concatenating q's unigram mean-pooling representation $h_{q_{1\_gram}} \in \mathbb{R}^{1\times d}$, 2-gram mean-pooling representation $h_{q_{2\_gram}} \in \mathbb{R}^{1\times d}$, word segmentation mean-pooling representation $h_{q_{seg}\in\mathbb{R}^{1\times d}}$, and mixed representation $h_{q_{mix}} \in \mathbb{R}^{1\times d}$, i.e.,

$$h_{q_{1\_gram}} = \text{mean-pooling}(h_{c_1}, \ldots, h_{c_m}), \tag{6}$$

$$h_{q_{2\_gram}} = \text{mean-pooling}(h_{c_1c_2}, \ldots, h_{c_{m-1}c_m}), \tag{7}$$

$$h_{q_{seg}} = \text{mean-pooling}(h_{t_1}, \ldots, h_{t_n}), \tag{8}$$

$$h_{q_{mix}} = h_{q_{1\_gram}} + h_{q_{2\_gram}} + h_{q_{seg}}, \tag{9}$$

$$h_q = \text{concat}(h_{q_{1\_gram}}, h_{q_{2\_gram}}, h_{q_{seg}}, h_{q_{mix}}), \tag{10}$$

where h, d, mean-pooling, and concat denote the word embedding, embedding size, average, and vertical concatenation operation.

We feed the abundant semantic representation h_q into the encoder of a transformer [45] to dynamically learn which semantic pattern is useful for matching items, and we regard the [CLS] token's representation as the query tower's representation $H_q \in \mathbb{R}^{1\times d}$, i.e.,

$$H_q = \text{Trm\_Encoder}^{CLS}(h_q). \tag{11}$$

3.2.2 Item Tower. For the item representation H_{item}, we use its title segmentation result $\mathcal{T} = \{t_1, \ldots, t_N\}$, and property set $\mathcal{P} = \{p_1, \ldots, p_M\}$ includes category, brand, and so on. We embed $\mathcal{T}$ and $\mathcal{P}$ into feature vectors $h_t \in \mathbb{R}^{1\times d}$ and $h_p \in \mathbb{R}^{1\times d}$, and then sum them to get the final representation $H_{item} \in \mathbb{R}^{1\times d}$. Mathematically,

$$h_t = \tanh\left(W_t \cdot \frac{\sum_{i=1}^{N} t_i}{N}\right), \tag{12}$$

$$e_{p_i} = W_{p_i} \cdot p_i, \tag{13}$$

$$h_p = \text{sum-pooling}(\{e_{p_i} | p_i \in \mathcal{P}\}), \tag{14}$$

$$H_{item} = h_t + h_p, \tag{15}$$

where W_t and W_{p_i} are the transformation matrices. Sum-pooling and tanh are the average operation and activation function. Applying a transformer encoder to capture the context of the title is not as effective as simple mean-pooling since the title is stacked by keywords and lacks grammatical structure [23].

3.2.3 Loss Function. We adapt the sampled softmax cross-entropy loss [20] as the training objective, achieving faster convergence and a better performance than other pointwise and pairwise losses. Unlike many toy datasets, in the e-commerce setting, the ground truth of relevant items is unknown, and there can be too many relevant products for a given query. For example, a query "红色连衣裙 (red dress)" has millions of relevant products. Note that our neural index is used for retrieving items of both relevance and user-preference to reduce information losses of truncation.

We use user feedback and a well-trained "query-item" relevance model [46] to select items for training. Specifically, given a query q, its positive samples i^+ are those that have been clicked in the past and are rated as relevant by the relevance model. Following [23], its negative samples i^- ($i_r^- \cup i_h^-$) include: (1) **r**andom negative samples i_r^- generated by sampling from the entire item pool I; and (2) **h**ard negative samples i_h^- generated by linear interpolation between i^+ and top-k cosine similarity scores with q of i_r^-'s representations. By examining the user logs, we obtains a train data set $\mathcal{D} = \{(q_1, i_1^+, i_1^-), \ldots, (q_N, i_N^+, i_N^-)\}$. The loss is defined by:

$$L(\nabla) = -\frac{1}{N} \sum_{q_j \in \mathcal{D}} \log \frac{\exp(\cos(q_j, i_j^+)/\tau)}{\sum_{i\in i_j^-} \exp(\mathcal{F}(q_j, i)/\tau)}, \tag{16}$$

where i and q denote the item tower's representation H_{item} and the query tower's representation H_q, respectively; τ is a temperature parameter we set to 2. Note that the negative samples i^- are generated during training. Similar to [23, 49], we use the same set of random negative samples for every q in the current batch to reduce computing resources. For convenience, we refer to our **M**ultiple **G**ranularities **S**emantic **I**ndexer as MGSI.

3.3 System Deployment

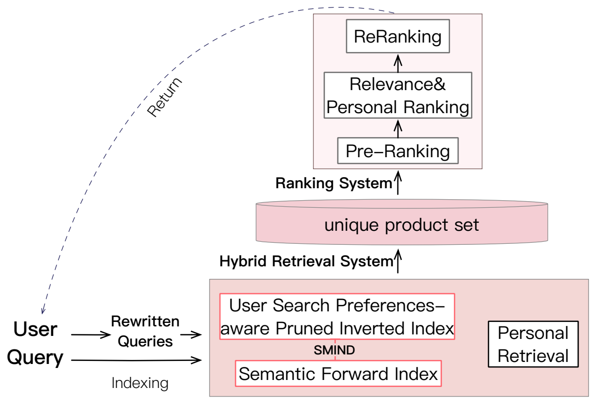


Figure 3: Overview of the architecture of the Taobao search engine with the proposed indexing system SMIND.

As shown in Figure 3, we deploy the proposed indexing system SMIND to the retrieval system of the Taobao search engine. When a user initiates a query, CLE-QR [22] will first rewrite it. We then use a hybrid retrieval system (SMIND and Personal Retrieval [7, 23]) to return a unique candidate set $\mathcal{I}$, which is then ranked by a large-scale deep-learning based ranking system. Finally, we expose the top-k products to users. See Section B of the Appendix for more details of deployment.

4 EXPERIMENTS

We introduce evaluation metrics, datasets, compared methods, implementation details, and offline and online quantitative and qualitative analyses of our method.

4.1 Evaluation Metrics

For offline evaluation, following [23], we use the metrics of Recall@K and P_{good}, which are defined as

$$Recall@K = \frac{\sum_{i=1}^{K} i_i \in T}{N}, \tag{17}$$

$$P_{good} = \frac{\sum_{i=1}^{K} \mathbb{I}(i_i)}{K}, \tag{18}$$

where $T = \{t_1, \ldots, t_N\}$ is the items clicked or purchased by the user u and $I = \{i_1, \ldots, i_K\}$ is the top-k items returned by an indexer. $\mathbb{I}(\cdot)$ is an indicator function. When item i is rated as good by our well-trained "query-item" relevance model (with an AUC of 0.92), its value is 1, otherwise 0.

For online evaluation, we consider the two most valuable commerce metrics, **GMV** (Gross Merchandise Volume) and **POC** (Pay Order Count). Additionally, we report four auxiliary system metrics, namely $\boldsymbol{Num_{rank}}$, $\boldsymbol{Recall_{clk}}$, $\boldsymbol{Recall_{pur}}$, and $\boldsymbol{P_{good}}$. See Section C of the Appendix for more details. Note that we also perform a Wilcoxon signed-rank test [10] to measure the significance of the difference. † and ‡ indicate that the improvement is statistically significant with $p < 0.05$ and $p < 0.01$, respectively.

4.2 Datasets

4.2.1 *Large-scale Industrial Offline DataSet.*

For offline evaluation, we randomly sample 1.5 million search interaction records in the next day. Following the online system's setting (Section 4.2.2), the size of the candidate item set of inverted index is about 2 billion and the semantic index is about 100 million.

4.2.2 *Online Dataset.*

We deploy a well-built SMIND in the Taobao search production environment to serve hundreds of millions of user query requests (Section 3.3). The size of the item candidate set is about 2 billion. Since the online inference of semantic index needs more computational resources, we have to reduce its item pool to the most active products, resulting in 100 million.

4.3 Compared Methods

4.3.1 *Offline Comparisons.*

Due to the differences in the item pools of the inverted indexer and the semantic indexer, we report the metrics of the two indexes separately.

Inverted Indexer. For the **In**Verted **I**ndexer, following [43], we report some strong baselines of static term-based pruning techniques, such as TF-based, TF-IDF based, and item quality Q-based. We also adapt DeepCT [8], SPARTA [50], and SPLADE [15, 16] for a fair comparison. Note that BM25 is an "query-item" score function and is not applicable in a static pruning setting where only single term posting lists can be accessed offline. The details are as follow:

IVI-TF The truncated posting list of each term is based on the scoring function $f(t, i)$, which is instantiated as term frequency (TF). Given a (term, item) pair, we get the TF value by counting the number of times the term t appears in the item title.

IVI-TF-IDF The scoring function $f(t, i)$ is instantiated as classical TF-IDF value.

IVI-Q For each term, we truncate its full item lists to a given length according to the item quality Q value.

IVI-DeepCT The value of the scoring function $f(t, i)$ is obtained from DeepCT [8].

IVI-SPARTA $f(t, i)$'s value is obtained from SPARTA [50].

IVI-SPLADE $f(t, i)$'s value is obtained from SPLADE [15, 16].

IVI-USP The scoring function $f(t, i)$ is instantiated by the Eq. (4), which considers the user search preferences.

IVI-USP-Q The proposed scoring function $f(t, i)$ (Eq. (5)), considering both user search preferences and item quality Q.

Semantic Indexer. For the **S**emantic **I**ndexer, our contribution is not to design any new model, but to combine query rewriting (QR) to facilitate its semantic matching, by addressing the mismatch problem mentioned earlier. Hence, we focus on the comparisons between MGSI and its combination with QR. Note that MGSI can be replaced by any semantic indexer mentioned in Section 2.2. The details are as follows:

MGSI The proposed **M**ultiple **G**ranularities **S**emantic **I**ndexer.

MGSI-QR In addition to triggering the MGSI with a user's original queries, we use the rewritten queries of *CLE-QR* [22].

4.3.2 *Online Comparisons.*

Our search engine is a complicated system since it has been tuned by hundreds of engineers and scientists for over ten years. We briefly introduce the our experimental environment for the readers' interest. For text matching, we have already deployed query/product tagging matching methods based on manual efforts and knowledge graphs [27] and the query rewriting technique CLE-QR [22]. For personal retrieval, we have graph-based methods [7, 13, 33] and an embedding-based method [23] online. We cannot simply remove them from the online system for comparisons. Hence, we only replace the indexer of text matching with SMIND. Since SMIND consists of IVI-USP-Q and MGSI, we report on two ablations, as follows:

SMIND The proposed hybrid indexer solution. It is used with both the user's original queries and rewrites of *CLE-QR.*

SMIND w/o IVIUSPQ We remove the user search preferences-aware pruned inverted indexer from SMIND.

SMIND w/o MGSI We remove the semantic indexer from SMIND.

4.4 Implementation Details

We collect search logs from the online Mobile Taobao App to build SMIND. Specifically, for the user search preferences-aware pruned inverted indexer, the purchase hypergraph $\mathcal{G}_{pur}$ is built from one month of (u, q, i) purchase interactions. The click hypergraph $\mathcal{G}_{clk}$ contains two weeks of (u, q, i) click interactions, about 10 billion records. The exposure hypergraph $\mathcal{G}_{expo}$ and relevance hypergraph $\mathcal{G}_{rel}$ contain one week (u, q, i) exposure and relevance interactions, about 350 billion records. The TermRank algorithm is implemented on our large-scale distributed system called Open Data Processing Service (ODPS).[1] Its number of iterations is set to 1 by cross-validation. The execution sequence of TermRank is: (1) we first

[1] https://www.aliyun.com/product/odps

initialize all $v(i_t)$ to 1, and run Eq. (3); and (2) we then run Eq. (2) to get the final value $v(i_t)$. The runtime of $\mathcal{G}_{pur}$, $\mathcal{G}_{clk}$, $\mathcal{G}_{expo}$, and $\mathcal{G}_{rel}$ is about 1 hour, 2h, 6h and 6h, respectively. Once we finish building the term-level search hypergraphs, users and items are static. To capture dynamic changes of users and products, we re-run TermRank periodically. Since our item set varies between days, we do it every day.

For training the semantic indexer, we use distinct query clicks/purchases of 7 consecutive days, a total of 4.7 billion records. The dimensions of the query tower and item tower are all set to 128. The batch size is 256. The setup of transformer encoder is 2 layers and 8 self-attention heads. All parameters are orthogonally initialized and learned from scratch. The training processes are run on the distributed TensorFlow platform [1] using 20 parameter servers and 100 GPU (Tesla P100) workers. We use the AdaGrad optimizer [12] with an initial learning rate of 0.1, since it improves the robustness of SGD for training large-scale networks [9]. We clip the gradient norm less than 3. The running time is about 48h.

4.5 Offline Experimental Results

4.5.1 Comparison of Pruned Inverted Indexer. As mentioned in Section 4.1, we report the metrics of $Recall@K$ and P_{good} on 1.5 million search interaction records. We set K to $1,000$.[2] As shown in Table 1, the performance of IVI-TF and IVI-TF-IDF are the worst in terms of the lowest metrics of $Recall@1000$ and P_{good}, which verifies that the search engine optimization (SEO) behavior of merchants make the TF and TF-IDF less effective.

Table 1: Offline performance of term-based pruned inverted indexers and ablation models.

Methods	$Recall@1000$	P_{good}
IVI-TF	0.8076	0.8739
IVI-TF-IDF	0.8099	0.8754
IVI-DeepCT [8]	0.8201	0.8766
IVI-SPARTA [50]	0.8199	0.8768
IVI-SPLADE [16]	0.8244	0.8773
IVI-SPLADEv2 [15]	0.8316	0.8790
IVI-Q	0.8192	0.8689
IVI-USP	0.8376	0.8812
IVI-USP-Q	**0.8510‡**	**0.8870‡**

We also use the value learnt by sparse retrieval models for pruning. Specifically, DeepCT [8], SPARTA [50] and SPLADE [15, 16] all outperform TF and TF-IDF, indicating that they do provide stronger term dependencies of textual relevance signal than TF and TF-IDF and mitigate the effect of SEO. The $Recall@1000$ score of IVI-USP is higher than that of the learned sparse models, which means that USP is more effective at capturing term dependencies than textual relevance and thus retrieve more user-preferred products.

However, learned sparse models frequently assign "wacky" weights to terms [28, 29, 31]. Especially in our case, since the user search logs are noisy (users may click/purchase items irrelevant to the query) [22, 23], it is hard to preserve high precision/relevance, which leads to a lower value of P_{good} than the proposed TermRank method. Finally, the proposed IVI-USP-Q outperforms all other methods in terms of $Recall@1000$ and P_{good}, indicating that considering both USP and the item quality factor Q helps to return more relevant items and better satisfy users.

[2]The Top-1000 items are obtained by our ranking system.

4.5.2 Comparison of Semantic Indexer. As discussed in Section 4.3.1, we here focus on the comparisons between MGSI and its combination with query rewriting (QR). As shown in Table 2, when combining the rewritten queries of QR with the semantic indexer, MGSI-QR achieves higher $Recall@1000$ scores, indicating that the retrieval performance of QR and semantic index can be mutually improved, by better addressing the vocabulary mismatch. The relevant retrieval ability of MGSI-QR seems to be weaker in terms of the P_{good} metric, as it is reduced by 0.85%. However, it does retrieve more user-preferred items than MGSI.

Table 2: Offline performance comparison between MGSI and MGSI-QR.

Methods	$Recall@1000$	P_{good}
MGSI	0.7876†	**0.7939‡**
MGSI-QR	**0.8219†**	0.7854‡

4.5.3 Qualitative Results of IVI-USP-Q. Table 3 shows the top 7 item categories and the corresponding number of products retained in the truncated posting list of the term "red" by the methods IVI-Q and IVI-USP-Q. With the fixed truncated length, 4 of the top 7 categories are consistent (i.e., Dress, Wool sweater, Lady bags and T-shirt), which shows that the item quality factor Q reflects the user search preference to a certain extent. However, due to different category preferences, the number of items in the four same categories is inconsistent. IVI-USP-Q considers both USP and Q, which makes the truncated item set closer to the general user search intention. Specifically, in the common categories, the number of items of "dress" and "lady bags" have increased by 141% and 67.7%, and these items could previously not be retained in "red"'s pruned posting list of IVI-Q due to a low Q value.

Table 3: With different term-based pruning methods, the top 7 item categories and their numbers within the truncated posting list of the term "红色 (red)."

term "红色(red)"			
IVI-Q		IVI-USP-Q	
Category	Num	Category	Num
毛针织衫 (Wool sweater)	2,776	**Dress**	6,140
连衣裙 (Dress)	2,552	Wool sweater	2,675
T恤 (T-shit)	2,465	**Lady bags**	2,231
运动鞋 (Sneaker)	1,575	T-shirt	2,229
衬衫 (Shirt)	1,440	单鞋 **(Shoes)**	2,208
女包 (Lady bags)	1,330	文胸 **(Bras)**	1,580
裤子 (Pants)	1,165	礼服 **(Canonicals)**	1,477

In the categories with differences, the number of items in the categories of "Shoes," "Bras," and "Canonicals" has increased significantly, and they have entered the top 7 of IVI-USP-Q from the

24-th, 28-th and 53-rd position of IVI-Q. The changes in these item categories and numbers are in line with our intuition, since when a user searches with term "red," he/she expects to see dress, lady bags, shoes, bras, and canonicals more. [3]

4.5.4 Case Study of Term Dependencies of USP. We use a small number of case studies to examine the term dependencies of USP. We randomly sampled 2 items of dress, and checked the value of each term $v(i_t)$ (i.e., Eq. (2)) in their title from the term-level user click hypergraphs $\mathcal{G}_{clk}$. A bigger value of $v(i_t)$ indicates that users more strongly prefer to use the term t to interact with item$_i$.

Item	Title
Item$_1$ PORTS	Chinese: ports(0.05) 宝姿(0.60) 裙子(0.01) 秋季(0.01) 时尚(0.00) 气质(0.06) 通勤(0.00) 修身(0.04) 收腰(0.02) 连衣裙(0.21) English: ports(0.05) 宝姿(0.60) skirt(0.01) autumn(0.01) fashion(0.00) temperament(0.06) commute(0.00) fit(0.04) waist(0.02) dress (0.21)
Item$_2$ self-portrait	Chinese: self–portrait(0.21) 水溶(0.09) 蕾丝(0.17) 连衣裙(0.49) 夏季(0.01) 薄款(0.00) 紧身(0.01) 泡泡袖(0.02) 中长裙(0.01) English: self–portrait(0.21) water soluble(0.09) lace(0.17) dress(0.49) summer(0.01) thin(0.00) tight(0.01) puff sleeves(0.02) midi skirt(0.01)

Figure 4: Case study of term dependencies of USP. Values in parentheses are calculated by TermRank.

As shown in Figure 4, although merchants perform SEO by stacking many textually relevant words, only a few keywords of each product will be searched by users. For item$_1$, the most important keywords are "宝姿" and "连衣裙 (dress)" since their values are bigger than others. Interestingly, item$_1$ has a shared Chinese and English keyword of the brand "PORTS," but most users tend to search it in Chinese. For item$_2$, its style is a puff sleeve lace dress, but the value of "puff sleeves" (0.02) $\ll$ "dress" (0.49). On the one hand, from the perspective of product textual relevance, "泡泡袖 (puff sleeves)" and "连衣裙 (dress)" are equally important to item$_2$. On the other hand, from the perspective of users, they rarely initiate the term "puff sleeve" to search for item$_2$. Moreover, the term "薄款 (thin)" is a textually relevant keyword of item$_2$ since it is item$_2$'s attribute. However, its value is 0, indicating that users have never interacted with item$_2$ using the term "thin." Instead, users use term "蕾丝 (lace)" more. The above examples show that the higher-order term dependencies of USP are crucial in e-commerce product search.

4.5.5 Dependencies of USP on Various Term-Level Hypergraphs. To further study the high-order term dependencies of user preferences, we show the value of each term in the title ($v(i_t)$) obtained

[3]The query "red" represents joy and happiness in Chinese customs. For example, at Chinese weddings, people wear red canonicals. Also, people give out red envelopes during Chinese New Year. Hence, people get used to associate the appearance of objects with the "red." More important, most of our users are women. Therefore, for the query "red," products of dresses, lady bags, and shoes should be more hits.

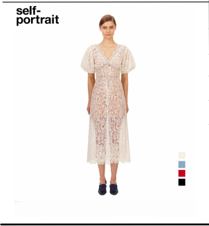
Item	Title Term	$\mathcal{G}_{pur}$	$\mathcal{G}_{clk}$	$\mathcal{G}_{expo}$	$\mathcal{G}_{rel}$
self-portrait 	self–portrait	0.44	0.21	0.12	0.10
	水溶(water soluble)	0.00	0.09	0.03	0.08
	蕾丝(lace)	0.22	0.17	0.09	0.07
	连衣裙(dress)	0.34	0.49	0.69	0.38
	夏季(summer)	0.00	0.01	0.03	0.04
	薄款(thin)	0.00	0.00	0.01	0.09
	紧身(tight)	0.00	0.01	0.01	0.06
	泡泡袖(puff sleeves)	0.00	0.02	0.01	0.08
	中长裙(midi skirt)	0.00	0.01	0.01	0.10

Figure 5: Comparison of term dependencies of USP on term-level hypergraphs. Values in the last four columns are calculated by TermRank.

from the term-level user search purchase/click/exposure/relevance preferences hypergraphs, $\mathcal{G}_{pur}$, $\mathcal{G}_{clk}$, $\mathcal{G}_{expo}$, and $\mathcal{G}_{rel}$, respectively.

As shown in Figure 5, given the title of a product, from user search relevance→exposure→click→purchase preferences, the importance of the title terms goes from divergence to convergence to a few specific terms. Specifically, in the user search relevance preferences hypergraph $\mathcal{G}_{rel}$, each term has a value indicating that each term in the title is relevant. In the exposure hypergraph $\mathcal{G}_{expo}$, the values of terms like "薄款 (thin)" and "紧身 (tight)," start to decrease and others are increasing. In the click hypergraph $\mathcal{G}_{clk}$, this phenomenon is more apparent. Finally, in the purchase hypergraph $\mathcal{G}_{pur}$, "self-portrait," "蕾丝 (lace)," and "连衣裙 (dress)" become key terms, while the others are vanishing. This means that when users purchase this product, they will only use the combination of terms "self-portrait," "蕾丝 (lace)," and "连衣裙 (dress)" to search. A comparison of the purchase hypergraph $\mathcal{G}_{pur}$ and the relevance hypergraph $\mathcal{G}_{rel}$ clearly tells us that the term dependencies in user search preference are far more important than textual relevance in product search.

4.6 Online Experimental Results

Beyond the offline experiments, we further conduct online evaluation to verify the effectiveness of `SMIND`.

4.6.1 Ablation Study. As mentioned in Section 4.1, for online performance analyses, we report the number of products considered in the ranking stage (i.e., Num_{rank}), the recall metric of click/purchase signal on the displayed item set I (i.e., $Recall_{clk}$ and $Recall_{pur}$), and the proportion of products with good relevance on I (i.e., P_{good}).

In addition to `SMIND`'s two ablation models (`SMIND w/o IVIUSPQ` and `SMIND w/o MGSI` discussed in Section 4.3.2, we also include the performance of the previous online baseline for comparison. Due to commercial confidentiality, we report relative values for some metrics.

Table 4: Online analyses of `SMIND` and its ablation models. The last three columns report relative values.

Methods	Num_{rank}	$Recall_{clk}$	$Recall_{pur}$	P_{good}
Online baseline	4351	–	–	–
`SMIND w/o IVIUSPQ`	4480	+2.68%	+1.32%	−0.45%
`SMIND w/o MGSI`	4552	+4.24%	+1.74%	**+1.62%**
`SMIND`	4651	**+5.45%**‡	**+2.81%**‡	+1.25%‡

As shown in Table 4, the two ablation modules can retrieve more item candidates that enter the ranking stage than the online baseline (higher Num_{rank}), indicating the effectiveness of the proposed indexer, IVI-USP-Q and MGSI, again. Specifically, MGSI (i.e., SMIND w/o IVIUSPQ) increases Num_{rank} by 3% and reduces P_{good} by 0.45%, showing that it improves the volume of user-preferred items as well as introducing some less relevant results. This is mainly because the personal ranking model scores "user-query-item" while the relevance model scores "query-item." There is a discrepancy between the two metrics. However, MGSI does improve the $Recall_{clk}$ and $Recall_{pur}$ by 2.68% and 1.32%, which demonstrates the dilemma of user preferences and uniform relevance criteria in e-commerce search. As for IVI-USP-Q (i.e., SMIND w/o MGSI), it improves all metrics, indicating that it retrieves more relevant and user-preferred products. Finally, since the hybrid indexer SMIND achieves the largest improvements, we conclude that its two ablation indexers complement each other and SMIND serves online users better.

Table 5: Online A/B test of SMIND on Mobile Taobao Search. The improvements are averaged over 18 days.

	GMV	POC	CTR	CVR
SMIND	+1.50%‡	+1.34%‡	+0.02$pt^{\ddagger}$	+0.02$pt^{\ddagger}$

4.6.2 *Online A/B Tests.*

Finally, we report the 18-day average of GMV and POC improvements (removing cheating traffic by the business analyst) achieved by SMIND. We include two additional metrics, click-through rate (CTR) and conversion rate (CVR).

Table 5 shows that SMIND improves GMV and POC by 1.50% and 1.34%, respectively, indicating that the daily increase income is hundreds millions. CTR and CVR also increase by 0.02pt and 0.02pt, respectively, which demonstrates that SMIND better satisfies users. We also deployed an unpruned inverted index for comparison, which can bring a marginal increase of 0.2% in GMV and POC at the cost of 50% increase in computing resources.

4.6.3 *User Query Shift Phenomenon.*

Because SMIND achieves significant improvements, we want to investigate its potential impact on users by examining users' search behavior versus their behavior with the previous online system. We first sort all queries in a descending order of their number of products recalled by the previous online system, and categorize them into 7 intervals, i.e., the smaller the number, the more recalled products. We refer to queries in the interval of No. 1–3 as top queries, those in the interval of No. 4, 5 as torso queries, and those in the interval of No. 6, 7 as tail queries. With the same online traffic volume setting, we then count the search frequency (SF) and item clicks (IC) in each interval.

Interestingly, as shown in Table 6, we find there is a user query shift phenomenon. Overall, since SMIND and the previous system have the same online traffic volume, their total SF are equal. But SMIND increases IC by 0.64%, indicating that its retrieved product collection better satisfies users, so that they are willing to click more. Let us take a closer look. The SF and IC ratio of the tail queries (No. 6–7) and the torso queries (No. 4–5) have increased significantly, while the top query (No. 1–3) decreased significantly. As the overall SF ratio of SMIND and previous system are equal, this indicates that there is a user query shift phenomenon. Specifically, in the previous retrieval system, when users initiate the torso/tail queries and fail to see the desired product set, they rewrite their own queries to top queries for viewing more products. However, on the test traffic, SMIND improves the matching ability of the previous indexing system, so that torso and tail queries can return more products that users are willing to click/purchase, which prevents users from rewriting queries to top queries and thus alleviating the Matthew effect [36] of user queries.

Table 6: The user search behavior with SMIND versus previous online system.

Query Type	No.	Search Frequency (SF)	Item Clicks (IC)
Overall	–	0.00%	+0.64%
Top	1	−2.76%	−2.18%
	2	−1.84%	−1.37%
	3	−0.96%	+0.01%
Torso	4	+1.69%	+2.89%
	5	+1.42%	+1.87%
Tail	6	+1.36%	+1.33%
	7	+1.62%	+1.11%

5 CONCLUSION

This paper presents an efficient solution for indexing billions of products, called **SM**art **IND**exing (SMIND), in Taobao search. SMIND addresses the challenge of reducing information loss during the static pruning of inverted indexes by integrating user search preferences (as discussed in Section 4.5.3) and mitigating the vocabulary mismatch problem inherent in term-based models (as outlined in Sections 4.5.2 and 4.6.1) that plagued the previous indexing system.

Our research demonstrates that understanding the term dependencies among users' search preferences is of paramount importance, surpassing textual relevance (as discussed in Sections 4.5.4, and 4.5.5). Meanwhile, we share the lessons learned from improving the performance of text matching, including the design of a pruning inverted indexer, a semantic indexer, its combination with a query rewriting technique, its effect on a large-scale commerce search system (Section 4.6.3), and its deployment scheme. Offline experiments and online A/B tests verify the effectiveness of SMIND.

We have successfully deployed the proposed system on Taobao Product Search, providing real-time services to hundreds of millions of users. In the future, we will incorporate term dependencies in user search preferences into the ranking stage to improve search personalization. Another aspect to investigate is the choice of semantic indexers. In particular, our semantic indexer was trained using only text corpora, whereas we also have rich image data. Therefore, we believe that building a multi-modal semantic indexer can further improve the performance of our system.

ACKNOWLEDGMENTS

The authors thank Prof. Qianli Ma and Ph.D. Qingwen Liu for their helpful discussions and comments on early drafts. We would also like to thank the anonymous reviewers for their helpful feedbacks.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A System for Large-scale Machine Learning. In *12th OSDI*. 265–283.

[2] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks* 30 (1998), 107–117.

[3] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient Query Evaluation Using a Two-level Retrieval Process. In *12th CIKM*. 426–434.

[4] David Carmel, Doron Cohen, Ronald Fagin, Eitan Farchi, Michael Herscovici, Yoelle S Maarek, and Aya Soffer. 2001. Static Index Pruning for Information Retrieval Systems. In *24th SIGIR*. 43–50.

[5] Wei-Cheng Chang, Daniel Jiang, Hsiang-Fu Yu, Choon Hui Teo, Jiong Zhang, Kai Zhong, Kedarnath Kolluri, Qie Hu, Nikhil Shandilya, Vyacheslav Ievgrafov, et al. 2021. Extreme Multi-label Learning for Semantic Matching in Product Search. In *27th SIGKDD*. 2643–2651.

[6] Xilun Chen, Kushal Lakhotia, Barlas Oguz, Anchit Gupta, Patrick Lewis, Stan Peshterliev, Yashar Mehdad, Sonal Gupta, and Wen-tau Yih. 2022. Salient Phrase Aware Dense Retrieval: Can a Dense Retriever Imitate a Sparse One?. In *Findings of the Association for Computational Linguistics: EMNLP 2022*. 250–262.

[7] Dian Cheng, Jiawei Chen, Wenjun Peng, Wenqin Ye, Fuyu Lv, Tao Zhuang, Xiaoyi Zeng, and Xiangnan He. 2022. IHGNN: Interactive Hypergraph Neural Network for Personalized Product Search. In *31st WWW*. 256265.

[8] Zhuyun Dai and Jamie Callan. 2020. Context-Aware Term Weighting For First Stage Passage Retrieval. In *43rd SIGIR*. 1533–1536.

[9] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *25th NeurIPS*. 1223–1231.

[10] Janez Demsar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *JMLR* 7 (2006), 1–30.

[11] Shuai Ding and Torsten Suel. 2011. Faster Top-k Document Retrieval Using Block-max Indexes. In *34th SIGIR*. 993–1002.

[12] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *JMLR* 12, 7 (2011), 2121–2159.

[13] Lu Fan, Qimai Li, Bo Liu, Xiao-Ming Wu, Xiaotong Zhang, Fuyu Lv, Guli Lin, Sen Li, Taiwei Jin, and Keping Yang. 2022. Modeling User Behavior with Graph Convolution for personalized product search. In *31st WWW*. 203–212.

[14] Marcus Fontoura, Vanja Josifovski, Jinhui Liu, Srihari Venkatesan, Xiangfei Zhu, and Jason Zien. 2011. Evaluation Strategies for Top-k Queries over Memory-resident Inverted Indexes. *Proceedings of the VLDB Endowment* 4, 12 (2011), 1213–1224.

[15] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From Distillation to Hard Negative Sampling: Making Sparse Neural IR Models More Effective. In *45th SIGIR*. 2353–2359.

[16] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *44th SIGIR*. 2288–2292.

[17] Jiafeng Guo, Yinqiong Cai, Yixing Fan, Fei Sun, Ruqing Zhang, and Xueqi Cheng. 2022. Semantic Models for the First-stage Retrieval: A Comprehensive Review. *ACM Transactions on Information Systems (TOIS)* 40, 4 (2022), 1–42.

[18] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based Retrieval in Facebook Search. In *26th SIGKDD*. 2553–2561.

[19] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning Deep Structured Semantic Models for Web Search Using Click through Data. In *22nd CIKM*. 2333–2338.

[20] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On Using Very Large Target Vocabulary for Neural Machine Translation. *arXiv preprint arXiv:1412.2007* (2014).

[21] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[22] Sen Li, Fuyu Lv, Taiwei Jin, Guiyang Li, Yukun Zheng, Tao Zhuang, Qingwen Liu, Xiaoyi Zeng, James Kwok, and Qianli Ma. 2022. Query Rewriting in TaoBao Search. In *31st CIKM*. 3262–3271.

[23] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. 2021. Embedding-Based Product Retrieval in Taobao Search. In *27th SIGKDD*. 3181—3189.

[24] Yiding Liu, Weixue Lu, Suqi Cheng, Daiting Shi, Shuaiqiang Wang, Zhicong Cheng, and Dawei Yin. 2021. Pre-trained Language Model for Web-scale Retrieval in Baidu Search. In *27th SIGKDD*. 3365–3375.

[25] Yiqun Liu, Kaushik Rangadurai, Yunzhong He, Siddarth Malreddy, Xunlong Gui, Xiaoyi Liu, and Fedor Borisyuk. 2021. Que2Search: Fast and Accurate Query and Document Understanding for Search at Facebook. In *27th SIGKDD*. 3376–3384.

[26] Xiaohui Long and Torsten Suel. 2003. Optimized Query Execution in Large Search Engines with Global Page Ordering. In *Proceedings 2003 VLDB Conference*. Elsevier, 129–140.

[27] Xusheng Luo, Le Bo, Jinhang Wu, Lin Li, Zhiy Luo, Yonghua Yang, and Keping Yang. 2021. AliCoCo2: Commonsense Knowledge Extraction, Representation and Application in E-commerce. In *27th SIGKDD*. 3385–3393.

[28] Joel Mackenzie, Andrew Trotman, and Jimmy Lin. 2021. Wacky Weights in Learned Sparse Representations and the Revenge of Score-at-a-Time Query Evaluation. *arXiv preprint arXiv:2110.11540* (2021).

[29] Joel Mackenzie, Andrew Trotman, and Jimmy Lin. 2023. Efficient Document-at-a-Time and Score-at-a-Time Query Evaluation for Learned Sparse Representations. *ACM Transactions on Information Systems* 41, 4 (2023).

[30] Alessandro Magnani, Feng Liu, Suthee Chaidaroon, Sachin Yadav, Praveen Reddy Suram, Ajit Puthenputhussery, Sijie Chen, Min Xie, Anirudh Kashi, Tony Lee, et al. 2022. Semantic Retrieval at Walmart. In *28th SIGKDD*. 3495–3503.

[31] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. Learning Passage Impacts for Inverted Indexes. In *44th SIGIR*. 17231727.

[32] Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. 2019. Semantic Product Search. In *25th SIGKDD*. 2876–2885.

[33] Xichuan Niu, Bofang Li, Chenliang Li, Rong Xiao, Haochuan Sun, Hongbo Deng, and Zhenzhong Chen. 2020. A Dual Heterogeneous Graph Attention Network to Improve Long-Tail Performance for Shop Search in E-Commerce. In *26th SIGKDD*. 3405–3415.

[34] Alexandros Ntoulas and Junghoo Cho. 2007. Pruning Policies for Two-Tiered Inverted Index with Correctness Guarantee. In *30th SIGIR*. 191–198.

[35] Debmalya Panigrahi and Sreenivas Gollapudi. 2013. Document Selection for Tiered Indexing in Commerce Search. In *6th WSDM*. 73–82.

[36] Matjaz Perc. 2014. The Matthew Effect in Empirical Data. *Journal of the Royal Society Interface* 11, 98 (2014), 20140378–20140378.

[37] Joseph Rocchio. 1971. Relevance Feedback in Information Retrieval. In *The Smart Retrieval System-experiments in Automatic Document Processing*. 313–323.

[38] Cristian Rossi, Edleno S. de Moura, Andre L. Carvalho, and Altigran S. da Silva. 2013. Fast Document-at-a-time Query Processing Using Two-tier Indexes. In *36th SIGIR*. 183–192.

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

[40] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. In *23rd WWW*. 373–374.

[41] Trevor Strohman, Howard Turtle, and W. Bruce Croft. 2005. Optimization Strategies for Complex Queries. In *28th SIGIR*. 219–225.

[42] Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. 2019. Ernie: Enhanced Representation Through Knowledge Integration. *arXiv preprint arXiv:1904.09223* (2019).

[43] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[44] Howard Turtle and James Flood. 1995. Query Evaluation: Strategies and Optimizations. *Information Processing & Management* 31, 6 (1995), 831–850.

[45] 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 *31st NeurIPS*, Vol. 30.

[46] Shaowei Yao, Jiwei Tan, Xi Chen, Keping Yang, Rong Xiao, Hongbo Deng, and Xiaojun Wan. 2021. Learning a Product Relevance Model from Click-Through Data in E-Commerce. In *30th WWW*. 2890–2899.

[47] Licheng Yu, Jun Chen, Animesh Sinha, Mengjiao Wang, Yu Chen, Tamara L Berg, and Ning Zhang. [n. d.]. CommerceMM: Large-scale Commerce Multimodal Representation Learning with Omni Retrieval. In *28th SIGKDD, pages=4433–4442, year=2022*.

[48] Han Zhang, Hongwei Shen, Yiming Qiu, Yunjiang Jiang, Songlin Wang, Sulong Xu, Yun Xiao, Bo Long, and Wen-Yun Yang. 2021. Joint Learning of Deep Retrieval Model and Product Quantization based Embedding Index. In *44th SIGIR*. ACM, 17181722.

[49] Han Zhang, Songlin Wang, Kang Zhang, Zhiling Tang, Yunjiang Jiang, Yun Xiao, Weipeng Yan, and Wen-Yun Yang. 2020. Towards Personalized and Semantic Retrieval: An End-to-End Solution for E-commerce Search via Embedding Learning. In *43rd SIGIR*. 2407–2416.

[50] Tiancheng Zhao, Xiaopeng Lu, and Kyusong Lee. 2020. SPARTA: Efficient Open-Domain Question Answering via Sparse Transformer Matching Retrieval. *arXiv preprint arXiv:2009.13013* (2020).

[51] Justin Zobel and Alistair Moffat. 2006. Inverted Files for Text Search Engines. *Comput. Surveys* 38, 2 (2006), 6–es.

A PROBLEM DEFINITION

For notation, let $U = \{u_1, \ldots, u_N\}$ be a set of N users, $Q = \{q_1, \ldots, q_L\}$ the user queries, $I = \{i_1, \ldots, i_M\}$ a collection of items, and $T = \{t_1, \ldots, t_K\}$ the unique terms that occur anywhere in the collection I.

A.1 Pruned Inverted Indexer

To cope with the large query loads with limited response latency, we prune the full inverted indexer at the posting level in advance. We retain all terms but truncate each term's posting list to a given length p. Given a term t in T, $I_{t\_F} = \{I_{t\_1}, \ldots, I_{t\_f}\}$ denotes its full posting list, and $I_{t\_P} = \{I_{t\_1}, \ldots, I_{t\_p}\}$ denotes its pruned posting list. Typically, for each term t, we keep the top-p item candidates from I_{t_F} based on the scores s between the term and items, i.e.,

$$s = \mathcal{T}(t, I_{t\_f}), \tag{19}$$

where $\mathcal{T}(\cdot)$ denotes the term-based scoring function and is instantiated with the TermRank algorithm.

A.2 Semantic Indexer

Whenever a user submits a query q, we would like to return a set of items $i \in I$ that are associated with that query. Typically, we predict the top-k item candidates from I based on the scores s between the query and items, i.e.,

$$s = \mathcal{F}(\phi(q), \psi(i)), \tag{20}$$

where $\mathcal{F}(\cdot)$, $\phi(\cdot)$, $\psi(\cdot)$ denote the scoring metric, query encoder, and item encoder, respectively. For large-scale retrieval, we use the two-tower neural model and instantiate $\mathcal{F}$ with the cosine function.

B ONLINE DEPLOYMENT

B.1 Retrieval Process Overview

As shown in Figure 1, when a user issues a query, the retrieval process generally involves five stages.

(1) It first rewrites the user query using a module called *Query Planner* (QP). For the query rewriting algorithms we used in *QP*, see [22].
(2) Next, the rewriting queries and original query are used to access the two indexers simultaneously. For the inverted index, two-tiered indexes are stored, I_P and I_F. I_F is the regular full posting list of a term while I_P is the truncated one with a given length. The semantic indexer comprises a query encoder neural network and item embedding indexes. The latter are exported by item encoder network in advance.
(3) During query evaluation by the inverted indexer, we use the term whose full posting list is the shortest within a query to access I_P, and the remainders access I_F. We call this strategy the *shortest posting list principle.* Then we traverse all the posting lists associated with the query and only return items that have all the keywords (AND-semantics) to ensure high search relevance (a.k.a. Document-At-A-Time strategy [44]). Besides, if the relevant results are less than a few hundreds, we will do query evaluation on all the I_F again. Hence, I_P greatly reduces both the index access and the scoring costs while it may degrade search quality.
(4) While visiting the semantic indexer, the query embeddings are predicted by the query encoder. And top-k results are returned by a maximum inner product search (MIPS) [21] operation between the query embedding and item embedding indexes.
(5) The results obtained from the two indexers are combined and duplicates are removed as the final candidates for subsequent ranking stages.

B.2 Indexer Deployment

Inverted Indexer. Due to the billion-scale item collection, we use a number of partitions (called *searchers*). Each contains an inverted index, with pruned and full index, on a subset of the items. The two-tiered indexes are created by an executor called *index building service*. Thus, the inverted indexer is deployed with distribution on machines of each *searcher*. Another kind of machines (called *query result service*) act as a query proxy that receives queries from the users. They broadcast queries to *searchers* for fetching the top-k results of each, and then merge the returned results into a distinct candidate set that is sent back to the following ranking.

Semantic Indexer. All the item embeddings are exported from the item tower by an *index building service*, and transmitted to an approximate near neighbor (ANN) indexing module.[4] They are also placed in multiple *searchers* because of their large number. Each partition of the ANN builds indexes of embeddings by hierarchical clustering (HC) algorithm with K-means and INT8 quantization to promote storage and search efficiency. The training sample size of HC is 4 million, and the max scan ratio is 0.01. The query tower is published to a *real-time prediction* platform for online inference. The mechanism of the *query result service* is simultaneously used for the semantic indexer. The final search results from `SMIND` are as follow:

```
(Inverted Indexer results) OR (Semantic Indexer results)
```

B.3 Serving Policy

The user search preferences-aware pruned inverted indexer serves all the query loads. For the semantic indexer, we propose to adapt an "offline&online" mixed serving strategy to reduce latency. We only feed the tail/new queries to online inference (*real-time prediction*). Despite the large number of tail and new queries, they account for about 20% of online traffic. The remaining 80% of online traffic is generated by millions of top queries, which typically remain unchanged for a period of time. Hence, we run the query tower of semantic indexer offline to export their embeddings, and then store them in an online key-value graph engine (*igraph*) for online ANN indexing.

C EVALUATION DETAILS

C.1 Offline Evaluation

Evaluation in industry product search is non-trivial since we are barely aware of the ground truth. We divide the evaluation into two parts, offline and online evaluation. Offline evaluation is based on sampled historical data, and online evaluation is based on daily new

[4]https://github.com/alibaba/proxima

data. The common metrics used by offline and online evaluation are Recall@K and P_{good}.

Recall@K is used for evaluating the performance since it has a positive correlation with the online income of Gross Merchandise Volume (GMV) metric [23]. Specifically, given a query q_u, the items clicked or purchased by the user u are regarded as the target set $T = \{t_1, \ldots, t_N\}$, and the top-k items returned by an indexer are regarded as the retrieval set $I = \{i_1, \ldots, i_K\}$. Recall@K is defined as:

$$Recall@K = \frac{\sum_{i=1}^{K} i_i \in T}{N}. \tag{21}$$

Precision@K is not suitable for evaluation when the ground truth is unknown. Instead, we use a well-trained "query-item" relevance model [46] (with an AUC of 0.92) to calculate the proportion of products with good relevance (abbreviated as good rate and denoted as P_{good}) in the retrieval set I, i.e.,

$$P_{good} = \frac{\sum_{i=1}^{K} \mathbb{I}(i_i)}{K}, \tag{22}$$

where $\mathbb{I}(\cdot)$ is an indicator function. When item i is rated as good by the relevance model, its value is 1, otherwise 0. Note that P_{good} is a user search experience-related metric.

C.2 Online Evaluation

Large-scale online tests are the most promising way to examine system performance since they provide an open environment to interact with users. We consider the two most valuable commerce metrics, **GMV** (Gross Merchandise Volume) and **POC** (Pay Order Count), which are defined as

$$GMV = \#\text{pay amount}, \tag{23}$$

$$POC = \#\text{pay order count}. \tag{24}$$

Besides GMV and POC, we report four auxiliary metrics, namely $\boldsymbol{Num_{rank}}$, $\boldsymbol{Recall_{clk}}$, $\boldsymbol{Recall_{pur}}$, and $\boldsymbol{P_{good}}$, which are defined as follows:

Num_{rank} Given a retrieval set $I = \{i_1, \ldots, i_K\}$, Num_{rank} is the number of items entering the ranking stage. This reflects the importance of an indexer in the search system.

$Recall_{clk}$ Eq. (21) is calculated on the item set I displayed to the users and the target set T is the items clicked by users.

$Recall_{pur}$ It is calculated similarly to $Recall_{clk}$, except that the target set T is the items purchased by users.

P_{good} Eq. (22) is calculated on the displayed item set I.