Generative Retrieval with Semantic Tree-Structured Identifiers and Contrastive Learning

Zihua Si
Renmin University of China
Beijing, China
zihua_si@ruc.edu.cn

Zhongxiang Sun
Renmin University of China
Beijing, China
sunzhongxiang@ruc.edu.cn

Jiale Chen
Kuaishou Technology Co., Ltd.
Beijing, China
chenjiale@kuaishou.com

Guozhang Chen
Kuaishou Technology Co., Ltd.
Beijing, China
chenguozhang@kuaishou.com

Xiaoxue Zang
Kai Zheng
Kuaishou Technology Co., Ltd.
Beijing, China
{zangxiaoxue,zhengkai}@kuaishou.com

Yang Song
Kuaishou Technology Co., Ltd.
Beijing, China
ys@sonyis.me

Xiao Zhang
Renmin University of China
Beijing, China
zhangx89@ruc.edu.cn

Jun Xu*
Renmin University of China
Beijing, China
junxu@ruc.edu.cn

Kun Gai
Independent
Beijing, China
gai.kun@qq.com

Abstract

In recommender systems, the retrieval phase is at the first stage and of paramount importance, requiring both effectiveness and *very high* efficiency. Recently, generative retrieval methods such as DSI and NCI, offering the benefit of end-to-end differentiability, have become an emerging paradigm for document retrieval with notable performance improvement, suggesting their potential applicability in recommendation scenarios. A fundamental limitation of these methods is their approach of generating item identifiers as text inputs, which fails to capture the intrinsic semantics of item identifiers as indices. The structural aspects of identifiers are only considered in construction and ignored during training. In addition, generative retrieval methods often generate imbalanced tree structures and yield identifiers with inconsistent lengths, leading to increased item inference time and sub-optimal performance. We introduce a novel generative retrieval framework named SEATER, which learns **SE**m**A**ntic **T**ree-structured item identifi**ER**s using an encoder-decoder structure. To optimize the structure of item identifiers, SEATER incorporates two contrastive learning tasks to ensure the alignment of token embeddings and the ranking orders of similar identifiers. In addition, SEATER devises a balanced k-ary tree structure of item identifiers, thus ensuring consistent semantic granularity and inference efficiency. Extensive experiments on three public datasets and an industrial dataset have demonstrated that SEATER outperforms a number of state-of-the-art models significantly.

CCS Concepts

• **Information systems** → **Recommender systems**.

Keywords

Recommendation; Generative Retrieval; Contrastive Learning

ACM Reference Format:
Zihua Si, Zhongxiang Sun, Jiale Chen, Guozhang Chen, Xiaoxue Zang, Kai Zheng, Yang Song, Xiao Zhang, Jun Xu, and Kun Gai. 2024. Generative Retrieval with Semantic Tree-Structured Identifiers and Contrastive Learning. In *Proceedings of the 2024 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region (SIGIR-AP '24), December 9–12, 2024, Tokyo, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3673791.3698408

1 Introduction

Modern recommendation systems (RS) predominantly use a two-stage retrieve-then-rank strategy [5]. During retrieval, a small subset of items (hundreds) is chosen from a vast item pool (millions). Considering the large scale of the entire pool, efficiency is vital for the retrieval model. Moreover, the success of the ranking model depends on the quality of retrieved items, highlighting the importance of retrieval effectiveness. Traditional models leverage dual-encoder architectures and Approximate Nearest Neighbor (ANN) algorithms. Initially, retrieval models represent users with a single vector [5, 13]. Subsequent studies [3, 16, 29] notice the inadequacy of single, finite-length vector representations, leading to the introduction of multi-vector retrieval. These approaches leverage multiple vectors to better express user interests and continue using ANN across multiple vectors for inference. However, the inner product of ANN theoretically requires a strong assumption for the Euclidean space, which may not be satisfied in practical applications. Hence, developing a model capable of capturing complex

*The corresponding author. Work partially done at Engineering Research Center of Next-Generation Intelligent Search and Recommendation, Ministry of Education.

SIGIR-AP '24, December 9–12, 2024, Tokyo, Japan

ACM ISBN 979-8-4007-0724-7/24/12
https://doi.org/10.1145/3673791.3698408

interactions, adequately representing user interests, and ensuring efficiency is a direction worth exploring.

To achieve this goal, tree-based indexing models like TDM [33] and JTM [32] from Alibaba have been introduced. They estimate interaction probabilities using deep models and achieve affordable efficiency by retrieving over tree-based indices, rather than the entire item pool. A recent trend in search [24, 26] views retrieval as a generation task. These models use transformer memory as a differentiable index and decode document IDs autoregressively as texts. The latest work TIGER [21] utilizes similar architectures for RS. They construct codebooks (so-called identifiers in this paper) to represent items and estimate interaction probabilities using the product probabilities predicted by transformers.

Despite their achievements, these generative models cannot meet the efficiency requirement for large-scale applications and their performance can be improved. Regarding effectiveness, these works treat identifier tokens purely as texts, optimizing with cross-entropy loss loosely related to the indexing structures, neglecting the inherent characteristics of such structures. The structural aspects are only considered when constructing identifiers and are not integrated into the loss function during training. In terms of efficiency, imbalanced tree-structured identifiers in DSI and NCI can result in increased and inconsistent inference time for items. The multiple transformer layers in TIGER increase the computational burden during inference.

To address such problems, we propose a generative model for the recommendation, namely SEATER, which learns **SE**m**A**ntic **T**ree-structured item identifi**ER**s via contrastive learning. We leverage an encoder-decoder model which encodes user interests and decodes probably the next items. The decoder represents items into equal-length identifiers with consistent semantics within the same level. SEATER assigns balanced k-ary tree-structured identifiers to items and learns semantics and hierarchies of identifier tokens through contrastive learning tasks. We construct such identifiers based on collaborative filtering information to incorporate prior knowledge. During training, we design two contrastive learning tasks to help the model comprehend the structure of item identifiers. Considering that each token represents an individual set of items, each identifier token has distinct semantics. The hierarchical relationship and inter-token dependency are inherent properties of such tree-structured indices. However, relying solely on user-item interactions for learning these complicated associations is challenging. It is necessary to introduce additional tasks to learn this structural information. We integrate two contrastive learning tasks in addition to the generation task. The first task employs the infoNCE loss, aligning token embeddings based on their hierarchical positions. The second task leverages a triplet loss, instructing the model to differentiate between similar identifiers. In this way, SEATER obtains both efficiency and effectiveness for item retrieval in RS. Extensive experiments across four datasets validate the effectiveness of the proposed model.

In summary, our main contributions are as follows:

- We introduce a generative framework, SEATER, for the retrieval phase of recommendation. We elaborate on the construction of identifiers, structural optimization based on contrastive learning.
- Utilizing two contrastive learning tasks, the model captures the semantics of the tokens and the hierarchies within the tree structure. Both tasks optimize identifiers' structures.
- The balanced k-ary tree structure ensures consistent semantic granularity for tokens at the same level and significantly reduces inference time compared with other tree-structured methods.
- Extensive experiments[1] on three public datasets and an industrial dataset have demonstrated that SEATER significantly outperforms several state-of-the-art (SOTA) methods, including dual-encoder, tree-based indexing, and generative methods.

2 RELATED WORK

Retrieval in Recommender Systems. In RS, the retrieval phase selects a subset of items from a vast corpus. For efficiency, the industry often uses dual-encoder models to represent users and items as vectors [3–5, 16, 18, 29]. And user preferences towards items are estimated through the inner product of vectors, which can be accelerated by ANN search for inference. The initial dual-encoder models represented users with a single vector [5, 9, 13]. Subsequent studies [3, 4, 16, 29], observed the limitations of expressing with a finite-length single vector and introduced multi-vector user interest modeling, continuing to utilize ANN search for inference. An alternate research direction aims to enable more intricate models with complex interaction estimation. TDM [33] and JTM [32] proposed by Alibaba involve tree-based indexing with advanced deep models, thereby facilitating more accurate estimation. RecForest [6] constructs a forest by creating multiple trees and integrates a transformer-based structure for routing operations. Similar to those studies, this paper seeks to retrieve items in a generative manner and optimize item identifiers from the indices perspective.

Generative Retrieval. In document retrieval, researchers have investigated using pre-trained language models to generate various types of document identifiers. For example, DSI [24] and NCI [26] utilize the T5 [20] model to produce hierarchical document IDs, while SEAL [2] (with BART [15] backbone) and ULTRON [31] (using T5) use titles or substrings as identifiers. AutoTSG [28] and NOVO [27], also based on T5, employ term-sets and n-gram sets as identifiers. There are studies exploring the identifier structures, such as GenRet [23] and LTRGR [17]. Generative document retrieval has been expanded to various fields. IRGen [30] uses a ViT-based model for image search, while TIGER [21] employs the T5-based architecture for RS. However, due to the *resource-intensive* nature of multiple transformer layers, these studies are ill-suited for large-scale item retrieval in RS. Different from them, this paper delves into the use of more *parameter-efficient* models for generative retrieval in such systems. Also, there are previous works utilizing the generative nature of language models for recommendation, including P5 [7, 10], TIGER [21], and GPTRec [19]. These approaches, after establishing item identifiers, also known as codebooks, do not optimize these identifiers' structures. Ideally, the models would optimize the identifiers related to corresponding indices. Towards this end, this paper learns the inherent hierarchy and relationships of identifier structures with the help of contrastive learning.

[1] Implementations available at this link (https://github.com/Ethan00Si/SEATER_Generative_Retrieval).

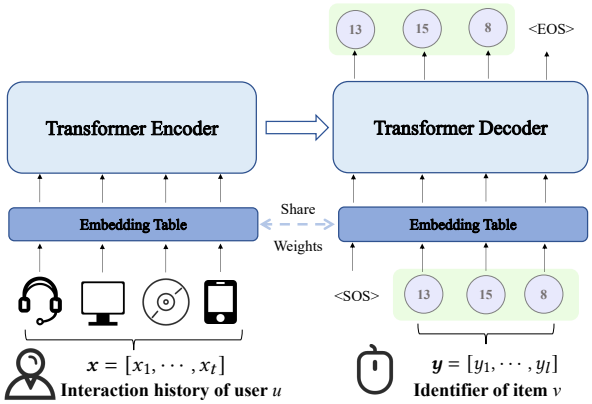

Figure 1: A brief illustration of SEATER. The retrieval model encodes the interacted items $\boldsymbol{x} = [x_1, x_2, \cdots, x_t]$ of user u and decodes the identifier $\boldsymbol{y} = [y_1, y_2, \cdots, y_l]$ of item v.

3 Method

In this section, we elaborate on the proposed model, detailing its model architecture, training, and inference stages.

3.1 Overview

Suppose a user $u \in \mathcal{U}$ accesses a retrieval system, and the system returns a list of candidates with each item $v \in \mathcal{I}$, where $\mathcal{U}$ and $\mathcal{I}$ denote the entire sets of users and items respectively. Let's use $\boldsymbol{x} = [x_1, \cdots, x_t] \in \mathcal{X}$ to denote the historically interacted items of user u. In generative retrieval, the identifier of each item $v \in \mathcal{I}$ is represented as a token sequence $\boldsymbol{y} = [y_1, \cdots, y_l] \in \mathcal{Y}$, where l is the length of the identifier. The goal of the generative retrieval model is learning a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$, which takes a user's interacted item sequence as input and generates a sequence of tokens (candidate identifiers).

As shown in Figure 1, the retrieval model feeds the user's behavior $\boldsymbol{x}$ into the encoder. Following this, the decoder employs an auto-regressive method to generate the item identifier $\boldsymbol{y}$ step by step. The probability of interaction between user u and item v is estimated as:

$$p(u, v) = \prod_{i=1}^{l} p(y_i | \boldsymbol{x}, y_1, y_2, \ldots, y_{i-1}) \tag{1}$$

where l denotes the length of item identifiers. To assign items with semantic representations, we convert all items into uniform-length identifiers, as depicted in Figure 2 (a). Identifier tokens capture item information from coarse to fine granularity, spanning from the beginning to the end. We use a multi-task learning approach to optimize both the model and identifiers, as depicted in Figure 2. The sequence-to-sequence task directs the model to generate valid identifiers, while the two contrastive learning tasks aid in grasping semantics and relationships among identifier tokens.

3.2 Retrieval Model

3.2.1 Encoder-Decoder Architecture. For the retrieval model, we employ the standard Transformer architecture [25]. Detailed Transformer structure specifics are omitted for brevity.

We leverage the Transformer encoder to capture user interests from behavior sequences:

$$\mathbf{X} = \text{Encoder}(x_1, x_2, \cdots, x_t), \tag{2}$$

where $\mathbf{X} \in \mathbb{R}^{t \times d}$ denotes the encoder hidden states of user interaction history $\boldsymbol{x} = [x_1, x_2, \cdots, x_t]$, t denotes the number of interacted items. The embeddings of t items serve as inputs to the encoder.

We exploit the Transformer decoder to model user-item interaction and predict the interaction probability in an auto-regressive manner. The decoder's hidden states are calculated as follows:

$$\mathbf{Y} = \text{Decoder}(\boldsymbol{x}, y_1, y_2, \ldots, y_l), \tag{3}$$

where $\mathbf{Y} \in \mathbb{R}^{l \times d}$ denotes the decoder hidden states of item identifier $\boldsymbol{y} = [y_1, y_2, \cdots, y_l]$, l is the length of the item identifier. The embeddings of identifiers serve as inputs to the decoder.

In our study, we refrained from stacking numerous Transformer layers, *e.g.,* 12 Transformer blocks in T5-Base [20]. We used **just one layer** to maintain efficiency in large-scale item retrieval contexts. In Section 5.5, we show that more layers indeed help the performance, with potential loss of efficiency.

The probability at step i can be modeled by softmax value of the i-th decoder hidden state $\mathbf{y}_i$ and candidate tokens C:

$$p(y_i | \boldsymbol{x}, y_1, y_2, \ldots, y_{i-1}) = \frac{\exp(\mathbf{y}_i^\intercal \mathbf{e}_{y_i})}{\sum_{y_{i'} \in C} \exp(\mathbf{y}_i^\intercal \mathbf{e}_{y_{i'}})}, \tag{4}$$

where $\mathbf{y}_i \in \mathbf{R}^d$ denotes the i-th vector in $\mathbf{Y} \in \mathbb{R}^{l \times d}$, $\mathbf{e}_{y_i} \in \mathbf{R}^d$ denotes the embedding for token y_i, and C is the set of all possible next tokens of size k given the prefix $[y_1, y_2, \ldots, y_{i-1}]$. This approach estimates interaction probability through product probabilities. The cross-attention mechanism and the decoder structure provide a comprehensive capture of interaction estimation beyond the inner product of dual-encoder models. Likewise, user interests are represented using a matrice $\mathbf{X}$ that incorporates the full historical sequence, as opposed to limited-length vectors. This method significantly improves the expressive power for user interests compared to the dual-encoder models.

3.2.2 Item Identifiers. Considering SEATER retrieves items using identifiers, the identifiers' construction is crucial. We have established a balanced tree structure to provide equal-length identifiers for the retrieval task, which offers numerous advantages.

SEATER utilizes a balanced k-ary tree structure to construct identifiers for items within set $\mathcal{I}$. To incorporate prior knowledge, we leverage a hierarchical clustering method with the constrained k-means [1] algorithm, to convert items into identifiers. Given an item set I to be indexed, we recursively cluster items into equal-size k groups until each group has fewer than k items. Detailed identifier tree construction can be found in Algorithm 1. We employ item embeddings $X_{1:N}$ extracted from trained SASREC [13] as the foundation for hierarchical clustering, leading to identifiers with collaborative filtering insights. We assign unique tokens for each clustered node because each node represents distinct item sets. We present a toy example to clarify our method. As shown in Figure 2 (a), a mouse, the 8-th item in set $\mathcal{I}$, is mapping into the identifier [*13, 15, 8*], where special tokens (start and end) are omitted. For instance, as shown in Figure 2 (a), token '8' representing item 8 and token '15' denoting items 7 and 8. Tokens' semantic granularity

Algorithm 1: Constructing equal-length identifiers.

Input: Item embeddings $X_{1:N}$, number of items N, number of branches k.
Output: Semantic item indexes $L_{1:N}$

Function ConstructIdentifiers(X) :
  # Min(Max) size of each cluster
  MinSize $\leftarrow \lfloor |X|/k \rfloor$, MaxSize$\leftarrow \lfloor |X|/k \rfloor + 1$
  $C_{1:k} \leftarrow$ Constrained-Kmeans(X, MaxSize, MinSize)
  $J \leftarrow$ empty list
  # Recursively clustering for each cluster
  For $i = 0$ **to** $k-1$ **do**
    $J_{current} \leftarrow [i] * |C_{i+1}|$
    if $|C_{i+1}| > k$ **then**
      $J_{rest} \leftarrow$ ConstructIdentifiers(C_{i+1})
    else
      $J_{rest} \leftarrow [0, \ldots, |C_{i+1}| - 1]$
    end if
    $J_{cluster} \leftarrow$ ConcatString($J_{current}, J_{rest}$)
    $J \leftarrow J$.AppendElements($J_{cluster}$)
  $J \leftarrow$ ReorderToOriginal(J, X, $C_{1:k}$)
  # Upon finishing clustering, assign unique IDs for each tree node
  if $|X| = N$ **then**
    $i \leftarrow N+1$, visited $\leftarrow$ empty dict, $L \leftarrow J$.copy()
    For $r = 1$ **to** N **do**
      # Each leaf node is encoded using its item ID
      $L_{r,\text{last column}} \leftarrow r$
      # Assign IDs to all non-leaf nodes
      For $l = 1$ **to** penultimate column **do**
        if $J_{r,1:l}$ **in** visited **then**
          $L_{r,\,l} \leftarrow$ visited($J_{r,1:l}$)
        else
          $L_{r,\,l} \leftarrow i$, visited($J_{r,1:l}$) $\leftarrow i$
          $i \leftarrow i+1$
  end if
  return L

differs by layer: the leaf layer conveys item-specific details, the penultimate layer captures information from a set of k items, and the topmost token embodies the whole item set's semantics. Thus, we allocate unique embeddings to individual tokens.

Formally, we embed all identifiers and items in an embedding table $\mathbf{E} \in \mathbb{R}^{M \times d}$, where M is the number of identifier tokens. In other words, *each tree node has unique token embedding*. Note that the identifier tokens at the leaf layer have a one-to-one correspondence with items. *We share embeddings between these tokens and items.* Given N as the item count, our identifiers add $(M-N)d$ extra embeddings compared to item embeddings. The fact $M - N \ll N$ causes an affordable increase in space overhead. See Section 4.2 for details.

Our construction method offers several distinct advantages: **(1)** All items are mapped into equal-length identifiers due to the balanced tree structure. The equal-length identifiers ensure that tokens at the same level possess consistent hierarchical semantics. In an imbalanced tree, an item's identifier might end at the third level while another extends to the fifth. This causes varied semantic granularity among third-level tokens. Furthermore, a balanced tree ensures shorter maximum identifier lengths (tree depth) than an imbalanced tree. This results in faster inference speed and equivalent processing time for all items, validated in Section 4.2. **(2)** We build the identifier tree with item embeddings from a different retrieval model. Using item embeddings informed by collaborative

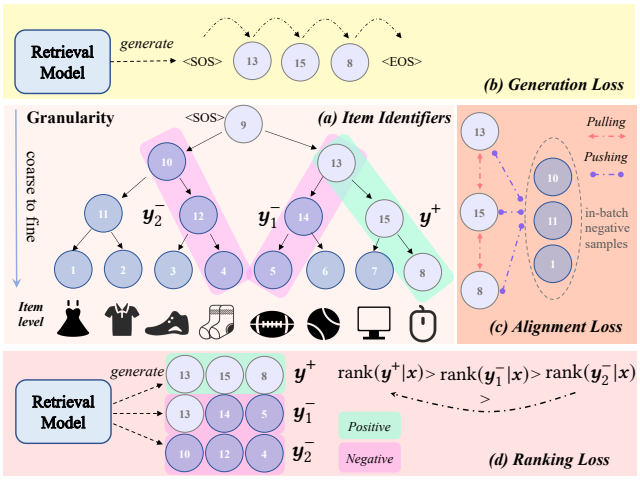

Figure 2: The proposed tree-structured identifiers and multi-task learning scheme. (a) An example of a balanced k-ary tree structure of item identifiers. Here k equals 2 for simplicity. In practice, k can be any integer ≥ 2. '9' denotes the start token. Each tree node corresponds to an unique token. (b)–(d) denote three losses for different tasks. (b) Generation Loss: guide the model to decode item identifiers. (c) Alignment Loss: grasp semantics and hierarchies of tokens. (d) Ranking Loss: differentiate between similar identifiers.

filtering, the identifiers effectively capture prior knowledge for recommendations. As shown in Figure 2 (a), items like 'mouse' and 'computer', which have similar user interactions, tend to have similar identifiers, *i.e.,* the prefix [*13, 15*]. Detailed empirical analyses in Section 5.4.1 have verified the strengths of our semantic identifiers. Importantly, using these embeddings doesn't add extra training loads. This aligns with industry practices where multiple models are used for multi-path retrieval, so there's no added training cost beyond SEATER. For instance, a company may simultaneously employ models like SASRec [13] and SEATER. Hence, using SASRec embedding does not necessitate additional overhead.

3.3 Training

Traditional recommendation models rely on user-item interaction data for training. In generative retrieval, where both the model and indices (item identifiers) are trained, we should also consider the indices' structure for training. To tackle this, we propose a multi-task learning scheme with two contrastive learning tasks and a generation task, as shown in Figure 2.

3.3.1 Generation Loss. We formulate the retrieval task as a sequence-to-sequence generation task for decoding item identifiers. To generate valid item identifiers, following [24, 26], we employ the sequence-to-sequence cross-entropy loss with teacher forcing. As depicted in Figure 2 (b), given a training sample $(\boldsymbol{x}, \boldsymbol{y})$, the loss function can be written as follows:

$$\mathcal{L}_{\text{gen}} = -\sum_{i=1}^{l} \log p(y_i | \boldsymbol{x}, y_1, y_2, \ldots, y_{i-1}), \tag{5}$$

where $\boldsymbol{x}$ denotes the user history, and $\boldsymbol{y}$ denotes the next interacted item's identifier.

3.3.2 Alignment Loss. Given that *each token has distinct semantics* and *inter-token relationships exist*, we utilize contrastive learning to learn identifiers from the indices perspective.

As depicted by the tree-structured identifiers in Figure 2, the parent token (*e.g.,* 15) encapsulates its child tokens (*e.g.,* 7 and 8), as one can only access the child tokens through the parent token. In tree-building, we group items into k clusters to form k child tokens. Thus, semantically, the parent token should align closely with the centroid of its child tokens. For instance, token 15 represents items 7 and 8, while token 14 represents items 5 and 6. Towards this end, we devise a contrastive learning objective shown in the Figure 2 (c). Given a token j, we employ the infoNCE loss to minimize the distance between it and its parent token p, while maximizing the distance between it and the in-batch negative instances:

$$\mathcal{L}_{\text{ali}} = -\log \frac{\exp(\cos(\mathbf{e}_j, \mathbf{e}_p)/\tau)}{\sum_{k \in \mathcal{B} \setminus j} \exp(\cos(\mathbf{e}_j, \mathbf{e}_k)/\tau)}, \tag{6}$$

where temperature τ is a hyper-parameter. The parent token p is viewed as the positive instance for token j. Other tokens in the same batch $\mathcal{B}$, excluding the child and parent of j, are viewed as negatives. This loss pulls the representations of tokens with parent-child relationships closer and pushes the representations of unrelated tokens apart.

3.3.3 Ranking Loss. The generative model compares candidate identifiers during inference (Section 3.4). Different identifier tokens index various items. In this way, the model needs to *discern subtle differences between similar identifiers*. We select identifiers with varying prefix lengths compared to the ground truth to guide the model in ranking them using a contrastive learning task. The varying prefix lengths imply the distinction of different identifiers within hierarchies.

For each ground truth identifier $\boldsymbol{y}^+$, we randomly sample q similar identifiers, denoted as $\boldsymbol{y}_1^-, \boldsymbol{y}_2^-, \cdots, \boldsymbol{y}_q^-$. For simplicity, in $\boldsymbol{y}_1^-$, the '1' denotes the first negative sample and not the first position of the identifier. We select q samples with q different shared prefix lengths from $\boldsymbol{y}^+$. These samples indicate related items with diverse similarity levels. For instance, as illustrated in Figure 2 (d) with $q = 2$, $\boldsymbol{y}_1^-$ shares one token with $\boldsymbol{y}^+$, whereas $\boldsymbol{y}_2^-$ shares none. In practice, we sample identifiers with more than two different tokens with $\boldsymbol{y}^+$. Then, we teach the model to rank these $q+1$ identifiers.

In specific, for each sample $(\boldsymbol{x}, \boldsymbol{y})$, we can get the representation vector of encoder hidden states $\mathbf{z}_x \in \mathbb{R}^d$ and decoder hidden states $\mathbf{z}_y \in \mathbb{R}^d$:

$$\mathbf{z}_x = \text{MEAN}(\mathbf{X}), \quad \mathbf{z}_y = \text{MEAN}(\mathbf{Y}), \tag{7}$$

where MEAN denotes the mean pooling, $\mathbf{X}$ and $\mathbf{Y}$ are obtained from equation 2 and 3 respectively. After obtaining hidden states, we rank different identifiers in pairs to instruct the model on the ranking order among these $q+1$ identifiers. The paired identifiers constitute the set Q, where $|Q| = C_{q+1}^2$. For any pair $(\boldsymbol{y}^\dagger, \boldsymbol{y}^\ddagger)$ in Q, we rank the sample with more identical prefix tokens with $\boldsymbol{y}^+$ higher. For example, as shown in Figure 2 (c), we rank $\boldsymbol{y}^+$ higher than $\boldsymbol{y}_1^-$, $\boldsymbol{y}^+$ higher than $\boldsymbol{y}_2^-$, and $\boldsymbol{y}_1^-$ higher than $\boldsymbol{y}_2^-$. In detail, we employ the triplet loss to steer the model toward learning the desired ranking orders:

Table 1: Time and space complexities analyses. We consider the beam search procedure for time complexity and the size of item identifiers for space complexity.

Models	Inference Time	Identifier Size
TDM [33]	$O(b \log_2 N)$	$O(Nd)$
RecForest [6]	$O(Tbk \log_k N)$	$O(Tkd)$
DSI [24]	$O(bkL)$	$O(kd)$
NCI [26]	$O(bkL)$	$O(kLd)$
SEATER	$O(bk \log_k N)$	$O(Nd)$

DSI&NCI: In the worst case, L equals $\frac{N}{k}$, resulting in $O(bkL)$ deteriorating to $O(bN)$. Empirically, L is $c \log_k N$, where c lies between 2 and 4, resulting in an inference time c times that of SEATER.
N: the number of items; k: the number of branches; d: the item embedding size; b: the beam size, L: the depth of tree in DSI and NCI; T: the number of trees in RecForest

$$\mathcal{L}_{\text{rank}} = \sum_{(\boldsymbol{y}^\dagger, \boldsymbol{y}^\ddagger) \in Q} \max\left\{0, s(\mathbf{z}_x, \mathbf{z}_{y^\dagger}) - s(\mathbf{z}_x, \mathbf{z}_{y^\ddagger}) + \xi\right\}, \tag{8}$$

where s is a similarity function, and ξ denotes a positive margin value. Here, we use $\boldsymbol{y}^\ddagger$ to denote the sample with more tokens in common with $\boldsymbol{y}^+$, and $\boldsymbol{y}^\dagger$ for the one with fewer. We set ξ as an adaptive value, $\xi = \beta * (\text{num}(\boldsymbol{y}^\ddagger) - \text{num}(\boldsymbol{y}^\dagger))$, to reflect rank differences in different pairs, where β is a hyper-parameter set to a small positive value, and $\text{num}(\boldsymbol{y})$ denotes the number of identical tokens between $\boldsymbol{y}$ and $\boldsymbol{y}^+$. The function s is defined as: $s(\mathbf{p}, \mathbf{q}) = \sigma(\mathbf{p}^T \mathbf{W}_s \mathbf{q})$, where σ denotes the sigmoid activation function, and the introduction of parameters $\mathbf{W}_s$ ensures the similarity estimation can be more flexible.

3.3.4 Multi-task Training. Finally, we train our model in an end-to-end manner under a multi-task learning scheme:

$$\mathcal{L} = \mathcal{L}_{\text{gen}} + \lambda_a \mathcal{L}_{\text{ali}} + \lambda_r \mathcal{L}_{\text{rank}}. \tag{9}$$

where λ_a and λ_r are hyper-parameters to balance different tasks. We also introduce L_2 regularization to avoid over-fitting, which is omitted here for conciseness.

3.4 Inference

In the inference phase, our objective is to extract the top n items from the entire candidate set. To achieve this, we employ a *constrained beam search* mechanism on the decoder module, specifically targeting tree-based identifiers, following NCI [26]. This ensures that the model's decoding aligns with the designated prefix tree, yielding valid identifiers.

4 Discussion

In this section, we compare SEATER with related previous work.

4.1 Comparison with Existing Work

Generative retrieval is an emerging research direction. We are at the forefront of incorporating the optimization of identifier structural information into the training phase of RS.

DSI [24] and NCI [26] pioneer in learning a generative model to map a string query to relevant docids for document retrieval.

They discover that tree-structured identifiers can establish structured information for candidate sets. TIGER [21], GPTRec [19], and P5 [7, 11] employ text, user-item interactions, or historical sequences as prior knowledge, utilizing distinct indexing methods, e.g., RQ-VAE and SVD. They consider the structure of identifiers in the construction phase, yet neglect it during the training process. The user-item interactions are insufficient for the model to learn complex structured information. SEATER optimizes structured information based on these findings. We construct a balanced tree to map items to equal-length identifiers, ensuring semantic consistency at each layer and enabling more efficient inference with reduced tree depth. We also introduce two contrastive learning tasks to the model training to aid in understanding the structure of the identifiers. Furthermore, SEATER achieves superior performance with just 1 transformer layer (with more parameters, SEATER can be better, as shown in Section 5.5), whereas previous works required multiple layers, such as TIGER with 4 layers and P5 with 6 layers.

4.2 Efficiency Analyses

We list complexity analysis of representative works with tree-structured identifiers in Table 1, validating that structures of item identifiers in SEATER are more efficient.

Regarding space complexity, our emphasis is on the storage cost of identifiers. Given that current recommendation systems inherently require storing an item embedding table of size Nd (N: the item count), our evaluation concentrates on the extra space introduced by identifiers. In SEATER, identifiers' leaf tokens share embeddings of corresponding items; only non-leaf tokens add to additional space overhead. Due to the structure of a balanced tree, the number of non-leaf tokens can be cumulatively calculated per layer: $1 + k + k^2 \cdots + \frac{\lceil \frac{N}{k} \rceil}{k} + \lceil \frac{N}{k} \rceil = \frac{k\lceil \frac{N}{k} \rceil - 1}{k-1}$. If N is the power of k, then $\frac{k\lceil \frac{N}{k} \rceil - 1}{k-1} = \frac{N-1}{k-1}$. In our experiments, k is set to 8 or 16. Consequently, the additional space cost $\frac{N-1}{k-1}d$ introduced by identifiers is significantly smaller compared to Nd (size of item embedding table).

To reduce time complexity, we leverage beam search during decoding. In real-world applications, intermediary encoder outputs in SEATER can be efficiently precomputed and stored, as shown in previous works. The bottleneck during inference is the beam search over identifiers. Compared to TDM's binary tree and RecForest's multiple trees, SEATER evidently shows an advantage in inference speed, as denoted in Table 1. Although DSI and NCI share a similar tree construction method with SEATER, their inference steps often amount to several times greater than SEATER. Due to their utilization of an imbalanced tree structure for identifiers, the max length of identifiers often is a constant multiple of $\log_k N$, and the max length critically influences inference speed. Therefore, SEATER demonstrates a superior inference time relative to other tree-based and generative models.

5 Experiment

5.1 Experimental Setup

We adhere to standard practices [3, 6, 29, 33] for item retrieval by choosing suitable datasets, baselines, and evaluation metrics.

Table 2: Statistics of three public and one industrial datasets.

Dataset	#Users	#Items	#Interactions	Density
Yelp	31,668	38,048	1,561,406	0.130%
News	50,000	39,865	1,162,402	0.050%
Books	459,133	313,966	8,898,041	0.004%
Micro-Video	0.75 million	6.1 million	85 million	0.002%

5.1.1 Dataset. Table 2 reports basic statistics of all the datasets. We have selected the following three public datasets: 1) **Yelp**[2]: This dataset is adopted from the 2018 edition of the Yelp challenge. The dataset encompasses business activities that occurred on the Yelp platform. 2) **Books**[3]: The Amazon review dataset [8] is one of the most widely used recommendation benchmarks. We adopt the 'Books' subset. 3) **News**[4]: The MIND dataset is a benchmark for news recommendation. It is collected from the behavior logs of the Microsoft News website. We adopt the 'MIND-small' subset.

To evaluate our model in a real-world situation, we collected an industrial large-scale dataset from a commercial app. 4) **Micro-Video**: we randomly selected 0.75 million users who used a micro-video app over two weeks in 2023. The historical behaviors have been recorded. Unlike other public datasets, this industrial dataset has not undergone any filtering and exhibits high sparsity that aligns with real industrial scenarios.

5.1.2 Evaluation Metrics & Protocal. Following the common practices [3, 6, 29], we divide each dataset into three parts, *i.e.,* training/validation/test sets by partitioning the users in a ratio of 8:1:1. For evaluation, we take the first 80% historical behaviors as context and the remaining 20% as ground truth. We strictly adhere to the evaluation framework in [3]. Please refer to [3] for details. As for metrics, we employ three widely used metrics, including *Hit Ratio* (HR), *Normalized Discounted Cumulative Gain* (NDCG)[5], and *Recall* (R). Metrics are calculated based on the top 20/50 recommended candidates (*e.g.,* HR@20). We calculated them according to the ranking of items and reported the average results.

5.1.3 Baseline and Implementation Details. We compare our model with SOTA models for item retrieval. The mainstream models commonly adopt a ***dual-encoder*** architecture: (1) **YoutubeDNN** (abbreviated as Y-DNN) [5]; (2) **GRU4Rec** [9]; (3) **MIND** [16]; (4) **ComiRec** [3] (the ComiRec-SA variant); (5) **SASREC** [13]; (6) **BERT4REC** [22]; (7) **Re4** [29]; We also include models with ***tree-based indexing*** : (8) **TDM** [33]; (9) **RecForest** [6]; Furthermore, we include latest ***generative*** recommendation models: (10) **GPTRec** [19] (the GPTRec-TopK variant); (11) **TIGER** [21] (implemented with 4 layers, the same as the original paper).

We also compare SEATER with **DSI** and **NCI**. Since DSI and NCI are primarily designed for search, leveraging textual query information as encoder input and built upon T5, they are not directly suitable for recommendation settings. SEATER's distinction with them stems from its approach to employing identifiers and the additional losses. To compare with DSI and NCI, we adapted

[2]https://www.yelp.com/dataset
[3]http://jmcauley.ucsd.edu/data/amazon/
[4]https://msnews.github.io/
[5]We compute the values based on the official definition of NDCG [12], while a few existing works do not. Details in https://github.com/Ethan00Si/SEATER_Generative_Retrieval/blob/main/Disccussion_of_NDCG.md

Table 3: Performance comparison on three public datasets and an industrial dataset. The best and the second-best performances are denoted in bold and underlined fonts, respectively. * indicates significant improvements with p-value < 0.05. In this table, SEATER uses a single layer of encoder-decoder. The performance of SEATER is further improved with more layers of encoder-decoder, as shown in Section 5.5.

Datasets	Metric	*Dual-encoder*							*Tree-based Indexing*		*Generative*		
		Y-DNN	GRU4Rec	MIND	ComiRec	SASREC	BERT4REC	Re4	TDM	RecForest	GPTRec	TIGER	SEATER
Yelp	NDCG@20	0.0412	0.0426	0.0414	0.0381	0.0466	0.0458	0.0362	0.0414	0.0434	0.0440	0.0539	**0.0572***
	NDCG@50	0.0613	0.0628	0.0611	0.0581	0.0699	0.0688	0.0595	0.0610	0.0646	0.0653	0.0769	**0.0810***
	HR@20	0.3366	0.3467	0.3502	0.3263	0.3711	0.3746	0.3263	0.3493	0.3503	0.3487	0.4087	**0.4201**
	HR@50	0.5396	0.5507	0.5409	0.5319	0.5799	0.5774	0.5468	0.5439	0.5512	0.5527	0.5922	**0.6118***
	R@20	0.0511	0.0529	0.0522	0.0508	0.0594	0.0590	0.0462	0.0524	0.0549	0.0559	0.0679	**0.0720***
	R@50	0.1045	0.1071	0.1046	0.1034	0.1206	0.1204	0.0976	0.1040	0.1110	0.1121	0.1271	**0.1353***
News	NDCG@20	0.0782	0.0836	0.0803	0.0753	0.0871	0.0829	0.0821	0.0830	0.0811	0.0813	0.0919	**0.0942**
	NDCG@50	0.1047	0.1114	0.1076	0.1011	0.1142	0.1072	0.1107	0.1067	0.1068	0.1065	0.1182	**0.1225***
	HR@20	0.3772	0.3872	0.3854	0.3738	0.3905	0.3640	0.3896	0.3821	0.3687	0.3731	0.4019	**0.4070**
	HR@50	0.5374	0.5480	0.5328	0.5279	0.5548	0.5210	0.5392	0.5248	0.5299	0.5305	0.5531	**0.5747***
	R@20	0.1192	0.1335	0.1282	0.1287	0.1383	0.1275	0.1292	0.1280	0.1270	0.1324	0.1408	**0.1456***
	R@50	0.2057	0.2287	0.2136	0.2163	0.2304	0.2099	0.2236	0.2080	0.2142	0.2182	0.2292	**0.2429***
Books	NDCG@20	0.0243	0.0192	0.0233	0.0250	0.0402	0.0352	0.0397	0.0235	0.0411	0.0271	0.0468	**0.0592***
	NDCG@50	0.0319	0.0260	0.0291	0.0331	0.0531	0.0457	0.0494	0.0330	0.0494	0.0373	0.0573	**0.0713***
	HR@20	0.0977	0.0820	0.0861	0.1169	0.1661	0.1374	0.1455	0.1101	0.1347	0.1181	0.1637	**0.2006***
	HR@50	0.1574	0.1354	0.1301	0.1788	0.2553	0.2124	0.2163	0.1832	0.1978	0.1962	0.2380	**0.2813***
	R@20	0.0447	0.0361	0.0402	0.0574	0.0793	0.0679	0.0712	0.0475	0.0625	0.0533	0.0766	**0.0972***
	R@50	0.0750	0.0626	0.0618	0.0890	0.1298	0.1088	0.1092	0.0849	0.0951	0.0938	0.1179	**0.1448***
Micro-Video	NDCG@20	0.0149	0.0202	0.0195	0.0211	0.0205	0.0197	0.0235	0.0201	0.0189	0.0187	0.0230	**0.0350***
	NDCG@50	0.0186	0.0254	0.0244	0.0289	0.0253	0.0238	0.0293	0.0240	0.0214	0.0221	0.0279	**0.0406***
	HR@20	0.1589	0.1991	0.1876	0.2198	0.2151	0.1951	0.2251	0.1980	0.1789	0.1877	0.2223	**0.2824***
	HR@50	0.2728	0.3287	0.3027	0.3567	0.3424	0.3159	0.3624	0.3077	0.2898	0.2928	0.3576	**0.4037***
	R@20	0.0118	0.0186	0.0175	0.0199	0.0191	0.0167	0.0231	0.0190	0.0178	0.0166	0.0211	**0.0310***
	R@50	0.0269	0.0383	0.0357	0.0403	0.0391	0.0338	0.0451	0.0342	0.0322	0.0331	0.0418	**0.0566***

SEATER by omitting the supplementary losses and adopting identifier structures from DSI and NCI. Detailed experimental results can be found in Table 4 and Section 5.3.

For baselines, we tune the hyper-parameters following the suggestions in the original papers. For SASREC and the five dual-encoder models, we train them using the sampled softmax loss [3], commonly adopted for the matching phase, setting the negative sample size to 1280. For other baseline models, we adopt the loss functions and training procedures described in the original papers. For all models, the dimension of item embeddings is set to 64. All the dual-encoder and transformer-based models make predictions based on ***brute-force*** retrieval, which involves calculating the probability over all items. For fair competition, we use the same item embeddings to build indexes for both SEATER and RecForest. Considering that TIGER requires using item text information to construct codebooks, and only the MIND dataset provides texts of items, we leveraged the SASREC embeddings for other datasets. As for TDM, RecForest, and SEATER, they predict based on ***beam search*** over item indexes, where we set the beam size to 50 for all of them.

We tune the hyper-parameters of SEATER as follows: the number of layers for encoder and decoder is set to 1; the values of loss coefficients, *i.e.,* λ_a and λ_r, are searched from [1e-2, 9e-2] with step 2e-2; the L_2 regularization weight is searched from [1e-4, 1e-5, 1e-6, 1e-7]; the number of tree branches k is searched in [2, 4, 8, 16, 32]; the number of sampled identifiers q is set to 4; the margin value β is searched in [0.01, 0.001, 0.0001]. We use Adam [14] for optimization with a learning rate of 0.001, and adopt the early stop training to avoid over-fitting. We provided code and data at an anonymous link (https://github.com/Ethan00Si/SEATER_Generative_Retrieval).

5.2 Overall Performance

Table 3 reports the overall performance on the four datasets. We have the following observations:

• **SEATER achieves the best performance on all datasets.** SEATER consistently outperforms baselines of various types by a large margin. Specifically, the relative improvements in R@50 on the Yelp, News, Books, and Micro-Video datasets are 6.45%, 5.43%, 11.56%, and 25.50%, respectively. These results underscore SEATER's effectiveness.

• **SEATER significantly outperforms dual-encoder models and tree-based indexing models.** Compared with dual-encoder models like SASREC, SEATER's improvement primarily stems from its generative decoding method, which models interaction probabilities more precisely than the inner product used by dual-encoder models. SEATER's improvement over models like ComiRec and Re4, which use multiple vectors to express user interests, confirms that expressing user interests through behavioral sequences provides a more comprehensive and thorough representation than using compressed vectors. Additionally, SEATER surpasses models employing

Table 4: Ablation study on four datasets. We assess the proposed two losses and the designed identifiers. Each loss contributes positively to the model, as shown in the middle four rows. Using DSI and NCI's decoder shows worse performance compared to SEATER's decoder structure, as shown in the last two rows.

Variants	Yelp			News			Books			Micro-Video		
	NDCG@50	HR@50	R@50	NDCG@50	HR@50	R@50	NDCG@50	HR@50	R@50	NDCG@50	HR@50	R@50
(0) SEATER	**0.0810**	**0.6118**	**0.1353**	**0.1225**	**0.5747**	**0.2429**	0.0713	**0.2813**	**0.1448**	**0.0406**	**0.4037**	**0.0566**
(1) w/o $\mathcal{L}_{\text{rank}}$ & $\mathcal{L}_{\text{ali}}$	0.0736	0.5920	0.1241	0.1142	0.5546	0.2309	0.0715	0.2770	0.1411	0.0390	0.3856	0.0514
(2) w/o $\mathcal{L}_{\text{rank}}$	0.0748	0.6029	0.1266	0.1201	0.5687	0.2386	0.0710	0.2802	0.1433	0.0394	0.3947	0.0531
(3) w/o $\mathcal{L}_{\text{ali}}$	0.0782	0.6054	0.1317	0.1167	0.5548	0.2335	**0.0721**	0.2779	0.1427	0.0395	0.3931	0.0534
(0) + w/o $\mathcal{L}_{\text{rank}}$ for negatives	0.0760	0.6063	0.1298	0.1208	0.5717	0.2401	0.0714	0.2807	0.1441	0.0402	0.3965	0.0544
(1) + DSI Identifiers	0.0551	0.5019	0.0952	0.0947	0.4862	0.1919	0.0408	0.1908	0.0902	0.0225	0.2727	0.0313
(1) + NCI Identifiers	0.0618	0.5316	0.1053	0.1047	0.5145	0.2067	0.0565	0.2128	0.1024	0.0343	0.3074	0.0365

contrastive learning to enhance user interest representation, such as Re4, validating the effectiveness of optimizing identifiers as indices.

• **SEATER surpasses other generative methods in overall comparisons.** TIGER utilizes 4 layers of encoder-decoder, while SEATER, with only 1 layer in this table, still achieves superior performance after significantly reducing resource consumption, validating the efficiency of SEATER. Moreover, on sparser and larger datasets, the improvement of SEATER is larger. The results indicate SEATER is better suitable for industrial applications. The improvement over TIGER and GPTRec also validates the effectiveness of enhancing the structure of item identifiers in SEATER, i.e., the balanced tree structure and contrastive learning tasks for understanding structural information.

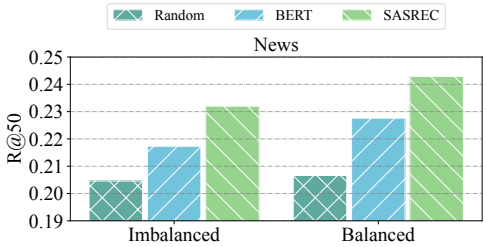

Figure 3: Different methods to construct identifiers. The collaborative filtering information and balanced structure make identifiers more informative.

5.3 Ablation Study

We evaluated the performance impact of SEATER's components via an ablation study. The results are reported in Table 4.

To assess the efficacy of the proposed losses $\mathcal{L}_{\text{rank}}$ and $\mathcal{L}_{\text{ali}}$, we test the following variants: Variant (1) excludes both loss terms; while Variants (2) and (3) remove $\mathcal{L}_{\text{rank}}$ and $\mathcal{L}_{\text{ali}}$ respectively, to study their contributions. Both loss functions demonstrate a favorable influence on the model performance. Removing either one individually leads to a decline in overall performance. Furthermore, we advanced our investigation by eliminating the ranking among negative samples within the ranking loss $\mathcal{L}_{\text{rank}}$, as shown in Equation 8. This led to the creation of Variant (0) + w/o $\mathcal{L}_{\text{rank}}$ for negatives. The performance of this Variant exceeds that of Variant (2) but falls short of Variant (0). This observation suggests that the inclusion of ranking among negative samples enhances the model's capability. These phenomenons illustrate that these two loss functions aid the model in comprehending the tree structure of identifiers, such as the inter-token relationships and hierarchies within the tokens.

To compare SEATER with DSI and NCI, we created two variants (1) + DSI Identifiers and (1) + NCI Identifiers, based on Variant (1). These variants leverage the imbalanced tree construction, token embedding allocation methods, and decoder structures from DSI and NCI. In specific, (1) + DSI Identifiers assigns k unique token vectors for a k-ary imbalanced tree. (1) + NCI Identifiers employs a layer-wise assignment of token embeddings which allocates kL unique token vectors for a k-ary imbalanced tree of depth L. (1) + NCI Identifiers also employs the PAWA decoder following the NCI paper. Apart from these, all other variables, such as the embedding used in building identifiers and the number of model layers, remain consistent with SEATER for a fair comparison. A significant decline in performance is noted in both variants compared with Variant (1), exhibiting an average performance drop exceeding 10%. This phenomenon validates the effectiveness of a balanced structure and suggests that shared embeddings for identifier tokens limit performance, especially in large-scale recommendation scenarios.

5.4 Study on Item Identifiers

5.4.1 Impact of Different Item Identifiers. To verify our statements in Section 3.2.2, we investigated the impact of tree balance and the utilization of different embeddings on model performance. As for tree balance, we utilized constrained k-means for a balanced tree and k-means for an imbalanced tree. As for embeddings used for hierarchical clustering, we explored employing SASREC's item embeddings, obtaining embeddings from items' textual descriptions using BERT, and randomly initialized embeddings. Owing to the exclusive presence of items' textual descriptions in the News dataset, we conducted experiments on this particular dataset. For BERT embeddings, we concatenated the category, subcategory, title, and abstract of the news articles within the News dataset to form the input for BERT. Subsequently, we extracted the embedding of the [CLS] token and employed it as the corresponding item embedding. For randomly initialized embeddings, we create them with random samples from a uniform distribution.

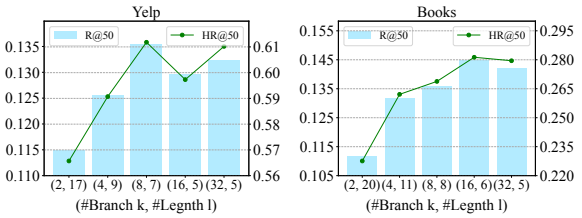

Figure 4: Impact of branch number k, ranging from 2 to 32, in terms of R@50 and HR@50. The corresponding identifier length l is also annotated.

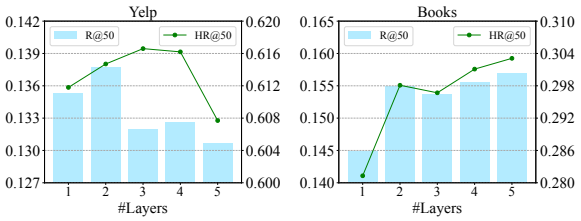

Figure 5: Analysis of the number of transformer layers.

The results are shown in Figure 3. We observed that utilizing a balanced tree yields performance improvements compared to using an imbalanced tree when employing identical sources of item embeddings. This is because a balanced tree ensures that tokens at the same level carry consistent semantic granularity, thereby capturing similar semantics within one layer. We also observed that employing the item embeddings from SASREC yields optimal performance. This is attributed to its alignment with collaborative filtering information, rendering it more suitable for recommendation tasks. Utilizing random identifiers yields the poorest performance, as it fails to impart any information gain to the identifiers.

5.4.2 Effect of Branch Number k. The variation in branch number k leads to a corresponding alteration in the length l of the item identifier. As k increases, l decreases. We adjusted the size of k and recorded the corresponding values of l along with the model's performance. This experiment employed two datasets, Yelp and Books, with varying item quantities. The results are illustrated in Figure 4. We observe that as k increases from 2 to 8 on the Yelp dataset (or 2 to 16 on the Books dataset), the model's performance reaches its peak, while further increasing k leads to a decline in performance. The performance improvement resulting from increasing k can be attributed to the reduction in identifier length l. As the beam search for inference cannot guarantee the selection of the correct next tokens at every step, a greater number of beam search steps (larger l) increases the probability of ultimate errors (due to cumulative errors). The decline in model performance as k increases from 8 to 32 on the Yelp dataset (or 16 to 32 on the Books dataset) is attributed to the fact that l remains relatively unchanged while k continues to increase. The beam search selects the top b options from $b * k$ candidate results at each step. The increase of k amplifies the difficulty of beam search at every step, while l results in a relatively unchanged total number of steps. Hence, both large and small values of k can lead to a decline in model performance.

5.5 Analysis on Parameter Count

We investigated the impact of the number of encoder-decoder layers. We observed distinct patterns across datasets of varying scales. The experiments were conducted on a small-scale dense dataset, Yelp, and a large-scale sparse dataset, Books.

As shown in the left part of Figure 5, when the number of layers increases from 1 to 3, the performance is further improved on the Yelp dataset. However, as the model's depth continues to increase, performance gradually deteriorates. We discovered that this phenomenon is attributed to the smaller scale of the Yelp dataset, where overfitting occurs as the model's depth increases. As shown in the right part of Figure 5, by increasing the model depth on the Books dataset, there is a continuous improvement in the model's performance. We posit that this is because a larger parameter count enhances the model's expressive capability on this dataset of a larger scale. We leave deeper models, such as those with 12 layers, for future work. Increasing the number of layers leads to a linear growth in computational complexity. This implies that the computational resources consumed by 2-layer and 3-layer models can be roughly considered as 2 times and 3 times that of a 1-layer model, respectively. Thus, considering the lower speed of deeper models, we find that 1-layer models can strike a satisfactory balance between performance and efficiency, as they have already attained peak performance compared with other baselines. Deepening the number of model layers is a promising direction for future research.

6 Conclusion

In this paper, we propose a generative retrieval model, namely SEATER, for recommendation. With contrastive learning tasks and balanced identifiers, SEATER achieves both efficiency and effectiveness by enhancing the structure of item identifiers. With the help of two contrastive learning tasks, SEATER captures the nuances of identifier tokens, including unique semantics, hierarchies, and inter-token relationships. Specifically, SEATER aligns token embeddings based on their hierarchical positions using the infoNCE loss and directs the model to rank similar identifiers in desired orders using the triplet loss. SEATER exploits a balanced k-ary tree structure for identifiers, leading to rational semantic space allocation and fast inference speed. This balanced structure maintains semantic consistency within the same level while different levels correlate to varying semantic granularities. Detailed analyses of time and space complexities validate the efficiency of the proposed model, enabling its application on large-scale retrieval. Extensive experiments on three public datasets and an industrial dataset verify that SEATER consistently outperforms SOTA models of various types.

Acknowledgments

This work was funded by the National Key R&D Program of China (2023YFA1008704), the National Natural Science Foundation of China (No. 62377044), Beijing Key Laboratory of Big Data Management and Analysis Methods, Major Innovation & Planning Interdisciplinary Platform for the "Double-First Class" Initiative, funds for building world-class universities (disciplines) of Renmin University of China, and PCC@RUC. Supported by Kuaishou Technology. Supported by the Outstanding Innovative Talents Cultivation Funded Programs 2024 of Renmin University of China.

References

[1] K.P. Bennett, P.S. Bradley, and A. Demiriz. 2000. *Constrained K-Means Clustering.* Technical Report MSR-TR-2000-65. 8 pages. https://www.microsoft.com/en-us/research/publication/constrained-k-means-clustering/

[2] Michele Bevilacqua, Giuseppe Ottaviano, Patrick Lewis, Wen tau Yih, Sebastian Riedel, and Fabio Petroni. 2022. Autoregressive Search Engines: Generating Substrings as Document Identifiers. In *arXiv pre-print 2204.10628*. https://arxiv.org/abs/2204.10628

[3] Yukuo Cen, Jianwei Zhang, Xu Zou, Chang Zhou, Hongxia Yang, and Jie Tang. 2020. Controllable Multi-Interest Framework for Recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2942–2951.

[4] Zheng Chai, Zhihong Chen, Chenliang Li, Rong Xiao, Houyi Li, Jiawei Wu, Jingxu Chen, and Haihong Tang. 2022. User-Aware Multi-Interest Learning for Candidate Matching in Recommenders. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Madrid, Spain) *(SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 1326–1335. https://doi.org/10.1145/3477495.3532073

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. New York, NY, USA.

[6] Chao Feng, Wuchao Li, Defu Lian, Zheng Liu, and Enhong Chen. 2022. Recommender Forest for Efficient Retrieval. In *NeurIPS*. http://papers.nips.cc/paper_files/paper/2022/hash/fe2fe749d329627f161484876630c689-Abstract-Conference.html

[7] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. 2022. Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5). In *Proceedings of the Sixteenth ACM Conference on Recommender Systems*.

[8] Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *Proceedings of the 25th International Conference on World Wide Web* (Montréal, Québec, Canada) *(WWW '16)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 507–517. https://doi.org/10.1145/2872427.2883037

[9] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1511.06939

[10] Wenyue Hua, Shuyuan Xu, Yingqiang Ge, and Yongfeng Zhang. 2023. How to Index Item IDs for Recommendation Foundation Models. *SIGIR-AP* (2023).

[11] Wenyue Hua, Shuyuan Xu, Yingqiang Ge, and Yongfeng Zhang. 2023. How to Index Item IDs for Recommendation Foundation Models. arXiv:2305.06569 [cs.IR]

[12] Kalervo Järvelin and Jaana Kekäläinen. 2000. IR Evaluation Methods for Retrieving Highly Relevant Documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Athens, Greece) *(SIGIR '00)*. Association for Computing Machinery, New York, NY, USA, 41–48. https://doi.org/10.1145/345508.345545

[13] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

[14] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[15] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703

[16] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipei Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. 2019. Multi-Interest Network with Dynamic Routing for Recommendation at Tmall. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 2615–2623. https://doi.org/10.1145/3357384.3357814

[17] Yongqi Li, Nan Yang, Liang Wang, Furu Wei, and Wenjie Li. 2023. Learning to Rank in Generative Retrieval. arXiv:2306.15222 [cs.CL]

[18] Fuyu Lv, Taiwei Jin, Changlong Yu, Fei Sun, Quan Lin, Keping Yang, and Wilfred Ng. 2019. SDM: Sequential Deep Matching Model for Online Large-Scale Recommender System *(CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 2635–2643. https://doi.org/10.1145/3357384.3357818

[19] Aleksandr V. Petrov and Craig Macdonald. 2023. Generative Sequential Recommendation with GPTRec. arXiv:2306.11114 [cs.IR]

[20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. http://jmlr.org/papers/v21/20-074.html

[21] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan H. Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. 2023. Recommender Systems with Generative Retrieval. arXiv:2305.05065 [cs.IR]

[22] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. ACM, New York, NY, USA, 1441–1450. https://doi.org/10.1145/3357384.3357895

[23] Weiwei Sun, Lingyong Yan, Zheng Chen, Shuaiqiang Wang, Haichao Zhu, Pengjie Ren, Zhumin Chen, Dawei Yin, Maarten Rijke, and Zhaochun Ren. 2023. Learning to Tokenize for Generative Retrieval. In *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 46345–46361. https://proceedings.neurips.cc/paper_files/paper/2023/file/91228b942a4528cdae031c1b68b127e8-Paper-Conference.pdf

[24] Yi Tay, Vinh Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Prakash Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. 2022. Transformer Memory as a Differentiable Search Index. In *NeurIPS*. http://papers.nips.cc/paper_files/paper/2022/hash/892840a6123b5ec99ebaab8be1530fba-Abstract-Conference.html

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762 http://arxiv.org/abs/1706.03762

[26] Yujing Wang, Yingyan Hou, Haonan Wang, Ziming Miao, Shibin Wu, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, Xing Xie, Hao Sun, Weiwei Deng, Qi Zhang, and Mao Yang. 2022. A Neural Corpus Indexer for Document Retrieval. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 25600–25614. https://proceedings.neurips.cc/paper_files/paper/2022/file/a46156bd3579c3b268108ea6aca71d13-Paper-Conference.pdf

[27] Zihan Wang, Yujia Zhou, Yiteng Tu, and Zhicheng Dou. 2023. NOVO: Learnable and Interpretable Document Identifiers for Model-Based IR. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management* (Birmingham, United Kingdom) *(CIKM '23)*. Association for Computing Machinery, New York, NY, USA, 2656–2665. https://doi.org/10.1145/3583780.3614993

[28] Peitian Zhang, Zheng Liu, Yujia Zhou, Zhicheng Dou, and Zhao Cao. 2023. Term-Sets Can Be Strong Document Identifiers For Auto-Regressive Search Engines. arXiv:2305.13859 [cs.IR]

[29] Shengyu Zhang, Lingxiao Yang, Dong Yao, Yujie Lu, Fuli Feng, Zhou Zhao, Tat-seng Chua, and Fei Wu. 2022. Re4: Learning to Re-Contrast, Re-Attend, Re-Construct for Multi-Interest Recommendation. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 2216–2226. https://doi.org/10.1145/3485447.3512094

[30] Yidan Zhang, Ting Zhang, Dong Chen, Yujing Wang, Qi Chen, Xing Xie, Hao Sun, Weiwei Deng, Qi Zhang, Fan Yang, Mao Yang, Qingmin Liao, and Baining Guo. 2023. IRGen: Generative Modeling for Image Retrieval. arXiv:2303.10126 [cs.CV]

[31] Yujia Zhou, Jing Yao, Zhicheng Dou, Ledell Wu, Peitian Zhang, and Ji-Rong Wen. 2022. Ultron: An Ultimate Retriever on Corpus with a Model-based Indexer. arXiv:2208.09257 [cs.IR]

[32] Han Zhu, Daqing Chang, Ziru Xu, Pengye Zhang, Xiang Li, Jie He, Han Li, Jian Xu, and Kun Gai. 2019. *Joint Optimization of Tree-Based Index and Deep Model for Recommender Systems*. Curran Associates Inc., Red Hook, NY, USA.

[33] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning Tree-Based Deep Model for Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 1079–1088. https://doi.org/10.1145/3219819.3219826