Unifying Generative and Dense Retrieval for Sequential Recommendation

Liu Yang
University of Wisconsin-Madison
AI at Meta[†]

Fabian Paischer
ELLIS Unit, LIT AI Lab, Institute for Machine Learning, JKU Linz, Austria
AI at Meta[†]

Kaveh Hassani, Jiacheng Li, Shuai Shao, Zhang Gabriel Li, Yun He, Xue Feng, Nima Noorshams, Sem Park, Bo Long
AI at Meta

Robert D Nowak
University of Wisconsin-Madison

Xiaoli Gao
AI at Meta

Hamid Eghbalzadeh *heghbalz@meta.com*
AI at Meta[‡]

Reviewed on OpenReview: *https://openreview.net/forum?id=jxdnFIsjCb*

Abstract

Sequential dense retrieval models utilize advanced sequence learning techniques to compute item and user representations, which are then used to rank relevant items for a user through inner product computation between the user and all item representations. While effective, these approaches incur high memory and computational costs due to the need to store and compare a unique embedding for each item–leading to lower resource efficiency. In contrast, the recently proposed generative retrieval paradigm offers a promising alternative by directly predicting item indices using a generative model trained on semantic IDs that encapsulate items' semantic information. Despite its potential for large-scale applications, a comprehensive comparison between *generative retrieval* and *sequential dense retrieval* under fair conditions is still lacking, leaving open questions regarding performance and resource efficiency trade-offs. To address this, we compare these two approaches under controlled conditions on academic benchmarks and observe performance gaps, with dense retrieval showing stronger ranking performance, while generative retrieval provides greater resource efficiency. Motivated by these observations, we propose LIGER (LeveragIng dense retrieval for GEnerative Retrieval), a hybrid model that combines the strengths of these two widely used approaches. LIGER integrates sequential dense retrieval into generative retrieval, mitigating performance differences between the two methods, and enhancing cold-start item recommendation in the evaluated datasets. This hybrid approach provides insight into the trade-offs between these approaches and demonstrates improvements in efficiency and effectiveness for recommendation systems in small-scale benchmarks.[1] [2]

[1] Code is available at `https://github.com/facebookresearch/liger`.
[2] †: Work done during internship at Meta., ‡: Corresponding author.

1 Introduction

Sequential recommendation methods (Kang & McAuley, 2018b; Zhou et al., 2020), have predominantly relied on advanced sequential modeling techniques (Hochreiter & Schmidhuber, 1997; Vaswani et al., 2017; Radford et al., 2019) to learn dense embeddings for each item and user. Using these embeddings, the most relevant items are retrieved through maximum inner product search. Despite their effectiveness, these approaches require computing inner products with every item in the dataset during retrieval stage, which can be computationally expensive as the number of items grows. Furthermore, each item must be represented by a unique embedding, which needs to be learned and stored, adding to the complexity.

In contrast, *generative retrieval* (Rajput et al., 2024) is an emerging approach, which deviates from the embedding-centric paradigm. Instead of pairwise comparisons between user and item embeddings, this approach utilizes a generative model to directly predict the item index. To better capture the sequential patterns within item interactions, items are indexed by "semantic IDs (SID)" (Lee et al., 2022a), which encapsulate their semantic characteristics. During the recommendation process, the model employs beam search decoding to predict the semantic ID of the next item based on the user's previous interactions. This method not only reduces the need for storing a number of embeddings that scales with the number of items, but also enhances the ability to capture deeper semantic relationships within the data. Additionally, adjusting the temperature during generation can naturally produce more diverse recommendations.

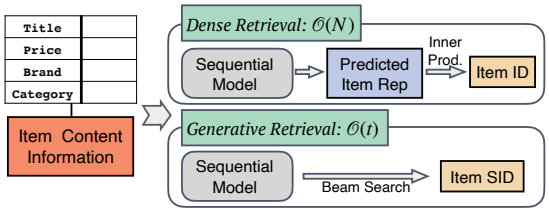
To distinguish the embedding-based retrieval approach from *generative retrieval*, which also leverages sequential information, we refer to it in this paper as *(sequential) dense retrieval*. The term *dense retrieval* is borrowed from the information retrieval domain (Tran & Yates, 2022), with "sequential" added to differentiate it from other dual-encoder-based architectures (Yi et al., 2019). The *generative retrieval* paradigm is well-positioned for future scaling in industrial recommendation systems (Singh et al., 2023; Zheng et al., 2025), offering significant savings in storage and inference time. However, while prior works continue to advance the dense retrieval paradigm (Hou et al., 2022c; Li et al., 2023b), generative retrieval methods are increasingly being integrated with pretrained models such as LLMs (Cao et al., 2024b; Paischer et al., 2024) to improve item recommendation.

Despite these advancements, there is a notable lack of direct comparisons under equivalent conditions, raising questions about which paradigm performs better given the same input information and how their performance balances against trade-offs in resource efficiency—such as computational and storage cost. Given the focus on academic benchmarks and limited resources, we narrow our study to small-scale datasets commonly used in research (Rajput et al., 2024; Cao et al., 2024a; Liu et al., 2024b; Li et al., 2023b; Hou et al., 2022c; Zheng et al., 2024). Within this context, we aim to explore these questions by comparing two established representative implementations of sequential generative and dense retrieval models, ensuring consistency in input information and experimental setups.

As shown in Figure 1, in our experiments, the sequential dense retrieval excels in ranking performance and cold-start item retrieval, albeit with a comparatively higher computational and storage footprint. In contrast, generative retrieval demonstrates lower performance–particularly on cold-start items–but may offer practical advantages in terms of model design and scaling flexibility. These observations are not entirely surprising, as dense retrieval methods have been well-established and extensively developed over time, excelling at learning high-quality embeddings that are particularly effective for recommendation tasks. Meanwhile, generative retrieval, as a relatively new paradigm, is still in its early stages of refinement. Despite its current limitations, generative retrieval holds significant potential (Singh et al., 2023; Zheng et al., 2025), and future improvements in architecture, training, or decoding strategies could further enhance its effectiveness.

At this stage, to harness the strengths of both paradigms, we propose a hybrid model, LIGER, that combines the resource efficiencies of generative retrieval with the robust embedding quality and ranking capabilities of dense retrieval. Our SID-based hybrid model applies dense retrieval to a limited set of candidates generated by a generative retrieval module, retaining the resource efficiency of generative retrieval while significantly improving performance, particularly for cold-start items. Specifically, our key contributions are as follows:

- We identify and analyze two primary observed limitations of the generative retrieval method on the small-scale academic benchmarks: (1) Generative retrieval exhibits a performance difference



Recall@10↑	Beauty		Sports		Toys		Steam	
	In-set	Cold	In-set	Cold	In-set	Cold	In-set	Cold
Dense	0.0931	0.0823	0.0553	0.0575	0.0952	0.0496	0.2011	0.0261
TIGER	0.0601	0.0	0.0382	0.0020	0.0578	0.0	0.1898	0.0
Perf Gap	0.0330	-	0.0171	-	0.0374	-	0.0113	-

Figure 1: *Performance Comparison Between the Implemented Generative and Dense Retrieval Methods Across Datasets.* Dense retrieval computes the inner product between predicted item representations and the entire item set, scaling with $\mathcal{O}(N)$ and requiring storage for $\mathcal{O}(N)$ embeddings. In contrast, generative retrieval stores only $\mathcal{O}(t)$ learnable embeddings and predicts the next item using beam search, scaling with $\mathcal{O}(tK)$, where K is the beam size and t is the number of Semantic IDs. Using identical item content information, both methods were trained on various datasets, and their performance, measured by Recall@10, is reported in the table on the right. While the implemented generative retrieval method reduces computational and storage costs, it shows lower performance compared to the implemented dense retrieval method in the datasets we evaluated.

compared to dense retrieval, given the same item information input, and (2) it tends to prioritize to items encountered during training, resulting in a lower probability of generating cold-start items due to under-representation.

- We propose LIGER (LeveragIng dense retrieval for GEnerative Retrieval), a novel method that synergistically combines the strengths of sequential dense and generative retrieval to significantly enhance the performance of generative retrieval. By integrating these methodologies, LIGER reduces the observed performance differences between dense and generative retrieval while improving the generation of cold-start items on the dataset we explored.

The remainder of the paper is organized as follows. In Section 2, we analyze the performance differences between the implemented generative and dense retrieval methods. Subsequently, we introduce the hybrid method LIGER in Section 3. The experimental results are presented in Section 4. Due to space constraints, we move the related work to Appendix A.

2 Analysis of Generative and Dense Retrieval Methods

In this section, we first introduce the generative retrieval (Rajput et al., 2024) and dense retrieval (Hou et al., 2022b) formula (see Section 2.1 and 2.2). Then in Section 2.3, we examine the performance difference between generative retrieval and sequential dense retrieval methods, and then discuss the challenges generative retrieval faces in handling item cold-start scenarios in Section 2.4.

2.1 Generative Retrieval Review

Generative retrieval approaches such as TIGER (Rajput et al., 2024) typically follow a two-stage training process. The first stage involves collecting textual descriptions for each item based on their attributes. These descriptions serve as inputs to a content model (e.g., a language encoder) that produces the text embeddings of the items, subsequently quantized by an RQ-VAE (Lee et al., 2022a) to attribute a semantic ID for each item. Formally, for each item $i \in \mathcal{I}$, we collect its p key-value attribute pairs $\{(k_1, v_1), (k_2, v_2), \ldots, (k_p, v_p)\}$ and format them into a textual description: $T_i = \text{prompt}(k_1, v_1, \cdots, k_p, v_p)$. The textual description T_i is then passed to the content model, yielding the text representation $\mathbf{e}_i^{\texttt{text}}$ for each item. We refer readers to Rajput et al. (2024); Lee et al. (2022a) for the training details of the RQ-VAE module. After the RQ-VAE is trained, we obtain the m-tuple semantic ID $(s_i^1, \cdots, s_i^m)$ for each item i. Notably, an m-tuple semantic ID, with a codebook size t, can theoretically represent t^m unique items.

In the second stage of training, the item text representation $\{\mathbf{e}_i^{\texttt{text}}\}$ and the trained RQ-VAE model are discarded, retaining only the semantic IDs. For each interaction history indexed by item IDs $\{i_1, i_2, \cdots, i_n\}$, the item IDs are replaced with their corresponding semantic IDs:

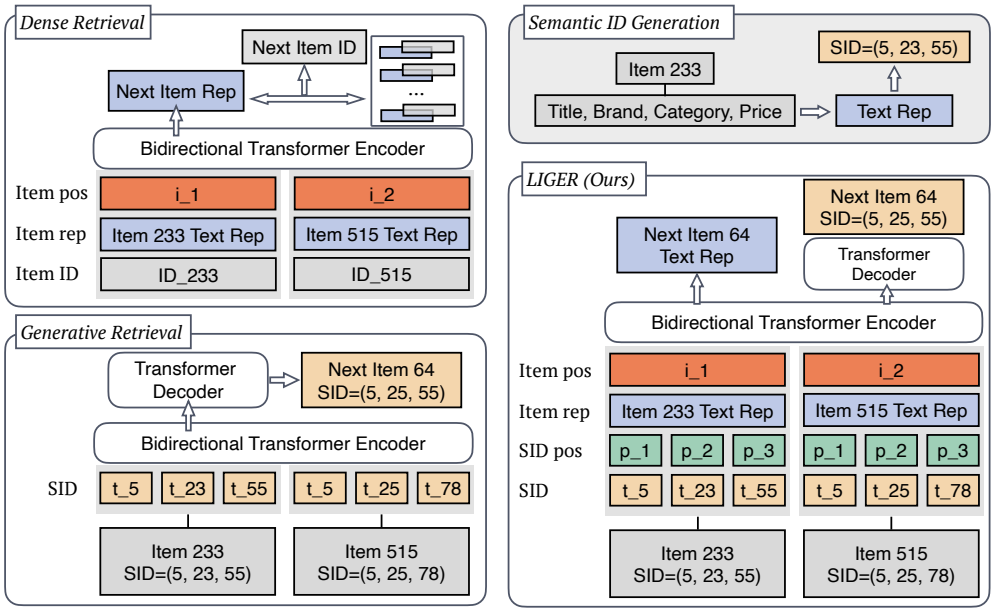


Figure 2: *Overview of Sequential Dense Retrieval, Generative Retrieval, and Our Hybrid Retrieval Method, LIGER.* Dense Retrieval (*upper left*) uses an encoder model to map item IDs and text representations into dense embeddings, which are used to predict the next item in the sequence based on similarity. Generative Retrieval (*lower left*) employs an encoder-decoder Transformer to generate the next item's semantic ID from the given semantic ID trajectory. These semantic IDs are derived from item features such as title, brand, price, and category (*upper right*). Our proposed Hybrid Retrieval, LIGER (*lower right*) combines both semantic ID input and item text representations, integrating dense and generative retrieval techniques. By taking item positions, text representations, and semantic IDs as input, and outputs both the predicted item embedding and the next item's representation.

$\{(s_1^1, s_1^2, \cdots, s_1^m), (s_2^1, s_2^2, \cdots, s_2^m), \cdots, (s_n^1, s_n^2, \cdots, s_n^m)\}$. Given the semantic IDs of the last n items a user interacted with, the Transformer is then optimized to autoregressively predict the next semantic ID sequence $(s_{n+1}^1, s_{n+1}^2, \cdots, s_{n+1}^m)$.

During inference, a set of candidate items is retrieved using beam search over the trained Transformer, selecting items based on their semantic IDs. A visual representation of the generative retrieval method is provided in Figure 2 *(Lower Left)*.

It is worth noting that although the item text representations are excluded from the second stage of training, they still contribute to the information utilized in LIGER since semantic IDs are derived based on the semantic similarities of item text representations. To ensure a fair comparison, we incorporate this information into the development of a comparable dense retrieval method in the following section, thereby accounting for the impact of these embeddings on the overall performance.

2.2 Sequential Dense Retrieval in Transductive Setting

Sequential dense retrieval methods typically consist of two main components: (1) learning item representations through sequence modeling, and (2) performing retrieval using dot-product search. To enhance the learning of item representations, several dense retrieval methods such as Hou et al. (2022b) employ an transductive setting, where item content information is integrated through text representation to enable transferable representation learning.

Building on these insights, we implement the dense retrieval method as follows: For each item i, we first obtain its text representation $\mathbf{e}_i^{\texttt{text}}$ using the procedure described in the first stage of Section 2.1. Additionally, we retrieve the learnable item embedding $\mathbf{e}_i = \texttt{Embd}(i)$ from the embedding table $\texttt{Embd}(\cdot)$ and obtain the positional embedding $\mathbf{e}_i^{\texttt{pos}}$ from a learnable positional embedding table. The input embedding for each item

is then computed as:

$$\mathbf{E}_i = \mathbf{e}_i + \mathbf{e}_i^{\texttt{text}} + \mathbf{e}_i^{\texttt{pos}},$$

and the sequence of input embeddings $\{\mathbf{E}_1, \mathbf{E}_2, \cdots, \mathbf{E}_n\}$ is provided to the Transformer. Given these n embeddings, the Transformer is trained to optimize the following objective:

$$\mathcal{L}_{\text{dense}}(\Theta, \{\mathbf{e}_i\}, \{\mathbf{e}_i^{\texttt{pos}}\}) = -\log \frac{\exp\left(\text{sim}(\hat{\mathbf{E}}(\Theta), \mathbf{e}_{n+1} + \mathbf{e}_{n+1}^{\texttt{text}})/\tau\right)}{\sum_{i \in \mathcal{I}} \exp\left(\text{sim}(\hat{\mathbf{E}}(\Theta), \mathbf{e}_i + \mathbf{e}_i^{\texttt{text}})/\tau\right)},$$

where $\hat{\mathbf{E}}(\Theta)$ represents the output embedding of the Transformer with parameter Θ, $\text{sim}(\cdot, \cdot)$ denotes the cosine similarity metric, τ is a temperature scaling factor, and $\mathcal{I}$ is the set of all items. Figure 2 *(Upper Left)* provides a detailed illustration of the dense retrieval model implementation.

2.3 The Observed Performance Difference

As detailed in Section 2.1, the generative retrieval model leverages both item text representations and sequential interaction information. In Section 2.2, we introduce sequential dense retrieval methods inspired by Hou et al. (2022c), designed to fully incorporate these sources of information in the training process.

To ensure a fair comparison between the generative retrieval and dense retrieval methods, we maintain consistency in model architecture, data pre-processing, and information utilization. The specific details of our experiment setup are described in Section 4.1 and in Appendix B.1. Due to the unavailability of TIGER's codebase, we made significant efforts to faithfully implement the TIGER method following the setup described in Rajput et al. (2024), conducting extensive hyperparameter tuning to optimize its performance, and configured the dense model's hyperparameters to align with the tuned TIGER settings. However these results, shown in Figure 1 *(Right)*, indicate a performance difference between the generative and dense retrieval methods in the datasets we evaluated.

There are notable differences between the two implemented methods: (a) **Different Item Indexing and Number of Embeddings**: As discussed in Section 2.1, representing N item requires the dense retrieval method to learn and store $\mathcal{O}(N)$ embeddings. In contrast, the semantic-ID-based generative retrieval method ideally only requires $\mathcal{O}(t)$ tokens, where $t^m \approx N$, with m being the length of the semantic ID tuple. However, it remains unclear whether the learned semantic ID can sufficiently capture the item's semantic meaning and effectively replace the item ID; (b) **Text Representation Input**: Dense retrieval utilize the item's text representation as additional input; and (c) **Prediction Mechanism**: Dense retrieval relies on maximum inner product search in the embedding space, whereas generative retrieval predicts the next item through next-token prediction via beam search.

To analyze the effect of (a), we modify the dense retrieval approach by replacing item IDs with semantic IDs while keeping all other components unchanged. This modified method is referred to as *Dense (SID)*. Formally, for each item i with semantic ID $(s_i^1, s_i^2, \cdots, s_i^m)$, we construct the input embedding for each semantic ID as:

$$\mathbf{E}_{s_i^j} = \mathbf{e}_{s_i^j} + \mathbf{e}_i^{\texttt{text}} + \mathbf{e}_i^{\texttt{pos}} + \mathbf{e}_j^{\texttt{pos}}, \tag{1}$$

where the $\mathbf{e}_{s_i^j}$ is the learnable embedding for each semantic ID: $\mathbf{e}_{s_i^j} = \texttt{Embd}(s_i^j)$, and $\mathbf{e}_j^{\texttt{pos}}$ is the positional embedding for each semantic ID. The final embedding for item i is then represented as: $\mathbf{E}_i = [\mathbf{E}_{s_i^1}, \mathbf{E}_{s_i^2}, \cdots, \mathbf{E}_{s_i^m}]$. During training, the cosine similarity between the predicted embedding and item's text representation is compared and maximized.

To examine the effect of (b), we augment TIGER with the item's text representation, referred to as *TIGER (T)*. Specifically, we use the same input as in Equation 1 and, during training, apply the next-token prediction loss on the semantic ID tuple of the next item.

The results in Figure 3 show that incorporating semantic IDs as input in dense retrieval (*Dense (SID)*) results in similar performance to the sequential dense retrieval (*Dense*), despite the difference in item indexing. This suggests that semantic indexing effectively preserves the information contained in item IDs, as it does not lead to a loss in retrieval performance. Moreover, supplementing TIGER with text representation as input (*TIGER (T)*) yields marginal improvements over TIGER alone; however, it still falls short of matching

the performance of dense retrieval. This suggests that the performance gap primarily stems from the third factor: a discrepancy in the objective function, as the next-token prediction loss is less effective for retrieval than direct similarity-based ranking.

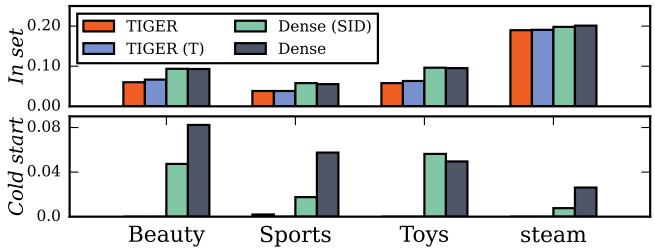


Figure 3: *Comparison of Recall@10 on in-set and cold-start dataset across various datasets.* Dense retrieval methods (both ID- and SID-based) exhibit robust performance in both in-set and cold-start settings. While *TIGER (T)* shows marginal improvements over TIGER, it still lags behind the performance of dense retrieval methods.

2.4 Challenges in Cold-Start Item Prediction with Generative Retrieval Models

Our investigation extends to the cold-start item generation problem, a critical issue in the dynamic environment of real-world recommendation systems. As new items are continuously introduced, they often lack sufficient user interactions, which impedes their predictability until a significant amount of interaction data is gathered.

For dense retrieval in the transductive setting (Hou et al., 2022c), the inclusion of item's text representations provide some prior information, thus partially retaining the ability to retrieve cold-start items, resulting in non-zero performance of cold-start item prediction shown in Figure 1 and Figure 3. In contrast, generative retrieval approaches such as TIGER and TIGER (T) fail to retrieve unseen items, as evidenced in the same figures. This highlights a fundamental limitation of generative retrieval methods, where the decoding process itself may hinder effective cold-start item generation.

To investigate this further, we analyze the generation probabilities of cold-start items in the Amazon Beauty dataset (details in Section 4.1) using a trained generative retrieval model and summarized the results in Figure 4 (b and c). Specifically, we ask the model to generate K candidates using beam search, which ranks items by their generation probabilities. Among these candidates, we define the minimum generation probability as p_K. Separately, we calculate the generation probability of the ground-truth cold-start item, denoted as $p^\star$. In Figure 4 (b), we present a histogram comparing $p^\star$ for all cold-start items with their corresponding p_K ($K = 10$). In Figure 4 (c), we vary $K \in \{10, 20, 40, 80\}$ to analyze its impact. The results show that cold-start items consistently have lower generation probabilities than the beam search threshold ($p^\star < p_K$), even when $K = 80$, effectively preventing them from being retrieved. Our findings indicate that the model's learned conditional probabilities tends to prioritize to items seen during training, leading to a significantly reduced capability to generate cold-start items.

It is worth noting that Rajput et al. (2024) propose an alternative solution to mitigate the issue of cold-start item generation. Their approach involves setting a predefined threshold ε for cold-start item within the retrieved candidate set of K items. This ensures that $K \cdot \varepsilon$ of the retrieved candidates are cold-start items by excluding other candidates that have higher generation probabilities. However, this method relies on prior knowledge of the ratio between recommended cold-start and non-cold-start items, which may not always be available. Therefore, we argue that the challenges in cold-start item generation persist for generative retrieval models, indicating a need for more robust solutions that do not depend heavily on predefined parameters or assumptions.

3 Methodology

The notion that "there is no free lunch" (Wolpert & Macready, 1997) holds true in the context of retrieval methods. As discussed in the previous section, a performance difference is observed between the generative retrieval and dense retrieval methods we implemented, with generative retrieval facing challenges in generating cold-start items. The improved performance of dense retrieval, however, comes with higher storage, learning, and inference costs. These costs are manageable in our current setup, where the candidate pool N is relatively small; however, they may become more significant at larger scales. On the other hand, genera-

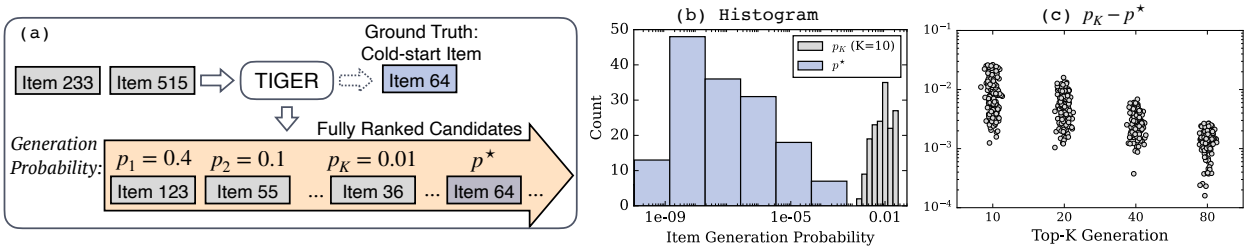


Figure 4: *TIGER Fails to Generate Cold-Start Items.* (a) The TIGER model generates a ranked list of candidates, with p_K denoting the generation probability of generating the K-th ranked item over all items. The ground-truth cold-start item has a generation probability of $p^\star$. (b) A histogram compares p_K (for $K = 10$) with $p^\star$ when the ground-truth item is cold-start, highlighting the disparity between them. (c) The difference $p_{\text{diff}} = p_K - p^\star$ is plotted for $K = 10, 20, 40, 80$. A successful generation of cold-start item occurs only when $p_{\text{diff}} \leq 0$, illustrating the model's limitations in handling cold-start items.

tive retrieval holds promise for more scalable and flexible retrieval—particularly in open-ended or large-scale scenarios (Rajput et al., 2024), but its performance lags behind in the datasets we evaluated.

Theoretical trade-offs between approaches are summarized in Table 1, where N represents the total number of items, t denotes the total number of semantic IDs, and K is the number of candidates to be retrieved during inference. Here, we denote the inference cost as the number of comparisons required for each method. We emphasize that this comparison is theoretical and abstracts away implementation-specific factors. In Section 4.4, we report empirical inference times for dense retrieval, generative retrieval, and our hybrid method; however, these measurements are not meant to account for system-level optimizations, hardware acceleration, or software engineering factors, which remain beyond the scope of this work.

Table 1: *Comparison of Dense Retrieval, Generative Retrieval, and Our Hybrid Methods LIGER Across Different Costs.* Here N represents the total number of items, t denotes the total number of semantic IDs used by generative retrieval method, and K indicates the number of candidates retrieved during inference.

	Dense Retrieval	Generative Retrieval	LIGER (Ours)
Learnable Embedding	$\mathcal{O}(N)$	$\mathcal{O}(t)$	$\mathcal{O}(t)$
(Fixed) Item Text Representation	$\mathcal{O}(N)$	-	$\mathcal{O}(N)$
Inference Cost	$\mathcal{O}(N)$	$\mathcal{O}(tK)$	$\mathcal{O}(tK)$
Cold-Start Item Generation	Yes	No	Yes

In this section, we propose a hybrid method, called LIGER, that combines the strengths of both approaches. Our goal is to improve upon the existing generative retrieval method: enabling it to generate cold-start items and bridging the gap with dense retrieval. To achieve this, we integrate text representations into the sequential model training phase of the generative retrieval method. The associated costs of LIGER are detailed in the last column of Table 1.

Formally, for each item i with semantic ID $(s_i^1, s_i^2, \cdots, s_i^m)$, we construct the input embedding for each semantic ID as defined in Equation 1. During training, the model is jointly trained on two objectives: the cosine similarity loss and the next-token prediction loss on the semantic IDs of the next item. The combined loss is formulated as:

$$\mathcal{L}(\Theta, \{\mathbf{e}_i\}, \{\mathbf{e}_i^{\text{pos}}\}, \{\mathbf{e}_j^{\text{pos}}\}) = -\log \frac{\exp\left(\text{sim}(\hat{\mathbf{E}}(\Theta), \mathbf{e}_{n+1}^{\text{text}})/\tau\right)}{\sum_{i \in \mathcal{I}} \exp\left(\text{sim}(\hat{\mathbf{E}}(\Theta), \mathbf{e}_i^{\text{text}})/\tau\right)} - \sum_{j=1}^{m} \log P(s_{n+1}^j \mid [\mathbf{E}_1, \cdots, \mathbf{E}_n]; \Theta).$$

The first term in the loss function ensures that the model learns to align the *encoder*'s output embedding with the text representation of the next item using a softmax over cosine similarity. The second term corresponds to the next-token prediction loss, where each token of the next item's semantic ID tuple is predicted sequentially in the *decoder*, conditioned on the historical input embeddings $[\mathbf{E}_1, \cdots, \mathbf{E}_n]$. Figure 2 (*Lower Right*) provides a detailed illustration of LIGER.

We adopt an *encoder-decoder* Transformer for our method. During inference, the decoder retrieves K candidate items using beam search. Given that cold-start items are relatively sparse compared to in-set items, we supplement the retrieved candidates with them to ensure their inclusion. The final set is then ranked using the encoder's output embeddings. This approach accounts for the periodic introduction of new items in recommendation systems, where cold-start items may otherwise be underrepresented. Details are shown in Figure 5 (*left*).

4 Experimental Setup and Results

In this section, we present the experimental results across various datasets and baseline methods, showcasing the performance on both in-set and cold-start items. Specifically, we assess the cold-start performance by testing on items that are unseen during training, which is determined by the dataset statistics.

4.1 Experimental Setup

Datasets. We evaluate LIGER on four datasets, preprocessing them using the standard 5-core filtering method (Zhang et al., 2019; Zhou et al., 2020). This process removes items with fewer than 5 users and users with fewer than 5 interactions. Additionally, we truncate sequences to a maximum length of 20, retaining the most recent items, following the setup in Rajput et al. (2024). Detailed statistics of the resulting datasets are provided in Appendix B.3.

- *Amazon Beauty, Sports, and Toys* (He & McAuley, 2016): We use the Amazon Review dataset (2014), focusing on three categories: `Beauty`, `Sports and Outdoors`, and `Toys and Games`. For each item, we construct embeddings by incorporating four key attributes: title, price, category, and description.

- *Steam* (Kang & McAuley, 2018b): The dataset comprises online reviews of video games, from which we extract relevant attributes to construct item embeddings. Specifically, we utilize the following attributes: title, genre, specs, tags, price, and publisher. To reduce the dataset size and make it more manageable, we apply subsampling by selecting every 7th sequence, thereby retaining a representative subset of the data.

When generating the item text representations, the item attributes are processed using the sentence-T5 model Ni et al. (2021) (XXL).

Semantic ID Generation. Utilizing the text representations generated from the sentence-T5 model, we employ a 3-layer MLP for both the encoder and decoder in the RQ-VAE (Lee et al., 2022a). The RQ-VAE features three levels of learnable codebooks, each with a dimension of 128 and a cardinality of 256. We use the AdamW optimizer to train the RQ-VAE, setting the learning rate at 0.001 and the weight decay at 0.1. To prevent collisions (i.e., the same semantic ID representing different items), following Rajput et al. (2024) we append an extra token at the end of the ordered semantic codes to ensure uniqueness.

Sequential Modeling Architecture and Training Algorithm. For the generative model, we utilize the T5 (Raffel et al., 2020) encoder-decoder model, configuring both the encoder and decoder with 6 layers, an embedding dimension of 128, 6 heads, and a feed-forward network hidden dimension of 1024. The dropout rate is 0.2. The dense retrieval model designed in Section 2.2 employs only the T5-encoder with 6 layers, while maintaining the same hyper-parameters. We use the AdamW optimizer with a learning rate of 0.0003, a weight decay parameter of 0.035, and a cosine learning rate scheduler. Additional details are presented in Appendix B.1.

Evaluation Metrics. We assess the model's performance using Normalized Discounted Cumulative Gain (NDCG)@10 and Recall@10. For dataset splitting, we adopt the leave-one-out strategy following (Kang & McAuley, 2018b; Zhou et al., 2020; Rajput et al., 2024), designating the last item as the test label, the preceding item for validation, and the remainder for training. During training, early stopping is applied based on the in-set NDCG@10 validation metric. For LIGER, which comprises two components: (A) the semantic ID prediction head and (B) the output embedding head, we implement early stopping based on the performance of component (B). To ensure fair evaluation of cold-start items, we exclude them from the RQ-VAE training to avoid data contamination.

Baselines. We compare our method against five state-of-the-art Item-ID-based dense retrieval methods: (a) SASRec (Kang & McAuley, 2018b); *feature-informed methods*, which incorporate additional item or user attributes beyond interaction history: (b) FDSA (Zhang et al., 2019), (c) S^3-Rec (Zhou et al., 2020); and *modality-based methods*, which leverage text as additional modal content to enrich item representations: (d) UniSRec (Hou et al., 2022b), (e) Recformer (Li et al., 2023a). Descriptions and implementation details for these baselines are provided in Appendix B.4. We also compare LIGER against TIGER (Rajput et al., 2024), a semantic ID-based generative retrieval method. Although subsequent works have built upon this paradigm using large language models (LLMs) (Zheng et al., 2023; Cao et al., 2024b), they rely on pre-trained LLMs, which are outside the scope of our comparisons.

4.2 Effect of Candidate Retrieval on LIGER

To demonstrate the efficacy of LIGER, we evaluate how its performance varies with the number of candidates K retrieved by the generative retrieval. For clearer insights, we present the normalized performance gap (NPG) in Recall@10 across various datasets. Specifically, let $r(K)$ denote the Recall@10 performance of LIGER with K candidates retrieved by generative retrieval, while $r_{\texttt{TIGER}}$ and $r_{\texttt{dense}}$ represent the Recall@10 performance of TIGER and dense retrieval, respectively. The NPG is then defined as:

$$\texttt{NPG}(K) = {}^{(r(K)-r_{\texttt{TIGER}})}\!/_{(r_{\texttt{dense}}-r_{\texttt{TIGER}})}.$$

Algorithm 1: Inference Process

Input : Interaction sequence $\{\mathbf{E}_1, \mathbf{E}_2, \ldots, \mathbf{E}_n\}$, Cold-start items $\mathcal{C}$, Beam size K
Output: Ranked list of items $\hat{\mathcal{I}}$
1. Beam search to retrieve top-K candidates: $\mathcal{I}_{\text{beam}} = \texttt{TF}([\mathbf{E}_1, \mathbf{E}_2, \ldots, \mathbf{E}_n]; K)$;
2. Combine with cold-start items: $\mathcal{I}_{\text{comb}} = \mathcal{I}_{\text{beam}} \cup \mathcal{C}$;
3. Rank Candidates with encoder's output $\hat{\mathbf{E}}$: $\hat{\mathcal{I}} = \texttt{topk}(\text{sim}(\hat{\mathbf{E}}, \mathbf{e}_i^{\texttt{text}}), \quad \forall i \in \mathcal{I}_{\text{comb}})$;

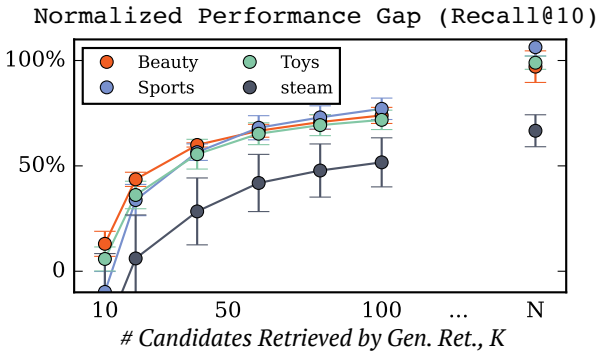


Figure 5: *Inference Process and LIGER's Performance in Bridging the Gap as the Number of Retrieved Candidates Increases.* The left panel illustrates the inference process of LIGER, detailing how candidate items are retrieved and ranked. In the algorithm, $\texttt{TF}(\cdot, K)$ denotes the Transformer generating K candidates using beam search based on the input sequence. The right panel shows the normalized performance gap between generative and dense retrieval models across several datasets (Beauty, Sports, Toys, Steam) for the in-set Recall@10 metric. In this normalization, 0% represents the performance of the generative retrieval model, while 100% corresponds to the performance of the dense retrieval model. The figure highlights how LIGER progressively bridges the performance gap as the number of candidates retrieved by the generative model increases.

We plot the NPG values for each dataset in Figure 5 (*right*) with varying K. The results show a consistent interpolation between TIGER and the dense retrieval approach: as the number of candidates K retrieved by the generative model increases, the likelihood of including the correct items in the candidate set grows. Consequently, LIGER progressively improves its performance on the Recall@10 metric, narrowing the gap with the dense retrieval method. In the next section, we will demonstrate the effectiveness of our method across various datasets and baseline methods.

4.3 Experimental Results

We present the results from the various datasets in Table 2, where the mean and standard deviation are calculated across three random seed runs. Traditional item-ID-based methods, such as SASRec exhibit poor in-set performance compared to semantic-ID-based models. However, when attribute information is included, models like FDSA and S^3-Rec show improved in-set performance. Nevertheless, their performance on cold-start items remains subpar due to the static nature of item embeddings. In contrast, models that utilize

Table 2: *Performance Comparison Across Baseline Methods on Amazon Beauty, Sports, Toys, and Steam Datasets.* The best performance is highlighted in bold, and the second-best performance is underlined. Our method consistently achieves either the best or second-best performance across all datasets, closely followed by modality-based baselines (UniSRec or RecFormer). We report LIGER's results where generative retrieval is used to retrieve 20 items, followed by the sequential dense retrieval.

	Methods	Inference Cost	**NDCG@10↑** (%)		**Recall@10↑** (%)	
			In-set	Cold	In-set	Cold
Beauty	SASRec	$\mathcal{O}(N)$	2.179 ± 0.023	0.0 ± 0.0	5.109 ± 0.042	0.0 ± 0.0
	FDSA	$\mathcal{O}(N)$	2.244 ± 0.135	0.0 ± 0.0	4.530 ± 0.357	0.0 ± 0.0
	S^3-Rec	$\mathcal{O}(N)$	2.279 ± 0.058	0.0 ± 0.0	5.226 ± 0.229	0.0 ± 0.0
	UniSRec	$\mathcal{O}(N)$	3.346 ± 0.057	1.422 ± 0.128	6.937 ± 0.110	3.704 ± 0.000
	RecFormer	$\mathcal{O}(N)$	2.880 ± 0.085	1.955 ± 0.433	6.265 ± 0.196	4.733 ± 0.943
	TIGER	$\mathcal{O}(tK)$	3.216 ± 0.084	0.0 ± 0.0	6.009 ± 0.204	0.0 ± 0.0
	LIGER (Ours)	$\mathcal{O}(tK)$	**4.020 ± 0.044**	**3.800 ± 0.523**	**7.447 ± 0.111**	**10.082 ± 1.285**
Sports	SASRec	$\mathcal{O}(N)$	1.160 ± 0.038	0.0 ± 0.0	2.696 ± 0.102	0.0 ± 0.0
	FDSA	$\mathcal{O}(N)$	1.391 ± 0.162	0.0 ± 0.0	2.699 ± 0.312	0.0 ± 0.0
	S^3-Rec	$\mathcal{O}(N)$	1.097 ± 0.033	0.0 ± 0.0	2.557 ± 0.034	0.0 ± 0.0
	UniSRec	$\mathcal{O}(N)$	1.814 ± 0.041	0.676 ± 0.244	3.753 ± 0.106	1.559 ± 0.447
	RecFormer	$\mathcal{O}(N)$	1.318 ± 0.053	1.797 ± 0.000	2.921 ± 0.167	3.801 ± 0.000
	TIGER	$\mathcal{O}(tK)$	1.989 ± 0.085	0.064 ± 0.056	3.822 ± 0.109	0.195 ± 0.169
	LIGER (Ours)	$\mathcal{O}(tK)$	**2.430 ± 0.075**	**2.731 ± 0.229**	**4.400 ± 0.127**	**5.848 ± 0.292**
Toys	SASRec	$\mathcal{O}(N)$	2.756 ± 0.079	0.0 ± 0.0	6.314 ± 0.178	0.0 ± 0.0
	FDSA	$\mathcal{O}(N)$	2.375 ± 0.277	0.0 ± 0.0	4.684 ± 0.483	0.0 ± 0.0
	S^3-Rec	$\mathcal{O}(N)$	2.942 ± 0.071	0.0 ± 0.0	6.659 ± 0.135	0.0 ± 0.0
	UniSRec	$\mathcal{O}(N)$	3.622 ± 0.056	1.090 ± 0.084	7.472 ± 0.058	2.477 ± 0.195
	RecFormer	$\mathcal{O}(N)$	3.697 ± 0.052	4.432 ± 0.094	**7.971 ± 0.170**	10.023 ± 0.516
	TIGER	$\mathcal{O}(tK)$	2.949 ± 0.049	0.0 ± 0.0	5.782 ± 0.163	0.0 ± 0.0
	LIGER (Ours)	$\mathcal{O}(tK)$	**3.756 ± 0.151**	**5.231 ± 0.531**	7.135 ± 0.244	**13.063 ± 0.516**
Steam	SASRec	$\mathcal{O}(N)$	14.763 ± 0.051	0.0 ± 0.0	18.259 ± 0.055	0.0 ± 0.0
	FDSA	$\mathcal{O}(N)$	8.236 ± 0.152	0.0 ± 0.0	14.773 ± 0.234	0.0 ± 0.0
	S^3-Rec	$\mathcal{O}(N)$	14.437 ± 0.127	0.0 ± 0.0	18.025 ± 0.222	0.0 ± 0.0
	UniSRec	$\mathcal{O}(N)$	-	-	-	-
	RecFormer	$\mathcal{O}(N)$	14.034 ± 0.123	0.120 ± 0.037	17.042 ± 0.275	0.319 ± 0.110
	TIGER	$\mathcal{O}(tK)$	**15.034 ± 0.064**	0.0 ± 0.0	18.980 ± 0.135	0.0 ± 0.0
	LIGER (Ours)	$\mathcal{O}(tK)$	14.951 ± 0.158	**0.512 ± 0.047**	**19.049 ± 0.234**	**1.466 ± 0.110**

text representations and pre-training, such as UniSRec and RecFormer, demonstrate enhanced capabilities in handling cold-start item scenarios. The inclusion of text embeddings during pre-training enables these models to better handle unseen items. TIGER, which is a semantic-ID-based generative retrieval model, outperforms item-ID-based methods in terms of in-set performance but still struggles with cold-start item generation.

Our model, LIGER, builds upon TIGER by using semantic-ID-based inputs and combining dense retrieval with semantic ID generation as outputs. This approach significantly improves upon the TIGER method and enables effective generation of cold-start items. Across all datasets, our method consistently achieves either the best or second-best performance, closely followed by modality-based baselines such as UniSRec and RecFormer. We adopt a hybrid approach for our reporting, where we use generative retrieval to retrieve 20 items and then rank them with cold-start items using dense retrieval. Comprehensive result including performance of LIGER with different number of retrieved items from generative retrieval is presented in Table 4. Additional ablation study on each component of LIGER is present in Appendix D.

4.4 Retrieval Efficiency at Scale: Dense vs. Generative

In this section, we evaluate and compare the inference cost of dense retrieval, generative retrieval (TIGER), and our hybrid method (LIGER) as the total number of candidate items increases. This analysis complements our earlier theoretical discussion in Section 3 by examining how these methods behave under both small-scale academic settings and large-scale scenarios where memory and compute constraints become criti-

cal. All experiments are conducted on a single NVIDIA RTX 4090 GPU with 24GB of memory, which allows us to test up to approximately 1.2 million items fully loaded into GPU memory.

As shown in Figure 6, under the academic benchmark we examined–where, after 5-core filtering, the total number of items is $N \approx 12{,}000$–dense retrieval (leftmost triangle) achieves faster inference than TIGER. This is primarily due to the highly optimized nature of inner product computations in dense retrieval, whereas beam search decoding in generative retrieval is less efficient in current implementations.

However, as we increase the number of items up to 1 million, we observe a clear linear relationship between inference time and the number of candidates N in dense retrieval. By fitting a linear trend to the observed runtime and extrapolating to $N \approx 16$ million, we estimate that the inference time would exceed 1000 seconds. In contrast, the generative retrieval module maintains a relatively constant runtime, as its decoding cost depends only on the number of semantic IDs generated. Under our current implementation, the semantic ID space supports up to $256^3 \approx 16$ million items, indicated by the orange dashed line in Figure 6.

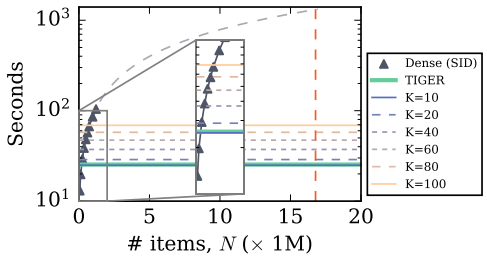


Figure 6: *Inference cost as a function of the total number of items.* We compare the inference time of dense retrieval with semantic IDs, generative retrieval (TIGER), and our hybrid model (LIGER) under varying values of K, as the total number of items increases. In theory, the semantic ID space used in our experiments can represent up to approximately 16 million items (indicated by the orange dashed line). In contrast, the inference cost of dense retrieval scales linearly with the number of items, surpassing that of generative methods once the candidate pool exceeds 1 million.

We also report the inference cost of our hybrid method, LIGER, across different values of K, where K denotes the number of candidates generated by the generative module and subsequently re-ranked by the dense module. As expected, the runtime increases approximately linearly with K; however–as shown previously in Figure 5–this increase is accompanied by notable performance gains. Overall, these results highlight a key trade-off: while dense retrieval is more efficient at smaller scales, generative retrieval offers better scalability in memory-limited settings.

5 Discussion

Addressing Cold-Start Items with Hybrid Retrieval Models. Generative retrieval method's struggle with cold-start items primarily stems from prioritizing familiar semantic IDs during training, as discussed in Section 2.4. To mitigate this issue, LIGER efficiently combines dense retrieval with generative retrieval. Specifically, LIGER first generates a small set of K candidates (where $K \ll N$) using generative retrieval, which is then augmented with the cold-start item set. This approach leverages the efficiency of generative retrieval to reduce the candidate pool while ensuring cold-start items are represented. The dense retrieval component further enhances cold-start performance by leveraging item text embeddings as prior information. As shown in Table 4, this integration ensures that when generative retrieval retrieves fewer than N items, the model maintains robust performance in cold-start scenarios, comparable to dense retrieval only.

Comparative Performance with Current Dense Retrieval Methods. While LIGER demonstrates competitive performance against existing baselines, its primary objective is to strike a balance between the generative and dense retrieval frameworks. As discussed in previous sections, there are observed performance differences between these two methods on the datasets we tested, even when using the same input information and model architecture. The results presented in this work aim to shed light on potential future directions for integrating these approaches, paving the way for the development of more robust and efficient recommendation systems.

Observed Performance Differences are Contextual to Small-Scale Datasets. We want to note that the performance differences observed in this study are influenced by various factors, including dataset size, implementation details, and the distribution of data collected under specific paradigms. Additionally, we are aware of industry-scale implementations of generative retrieval paradigms (Singh et al., 2023; Zheng et al.,

2025) that outperform dense retrieval approaches in real-world settings. In this work, our goal is not to assert a definitive performance gap between the two paradigms. Instead, we focused on aligning the two methods as closely as possible within the scope of academic benchmarks, which typically use small-scale datasets. Our findings are meant to provide insights specific to this academic setting and should not be extrapolated to large-scale, real-world applications without further investigation. Understanding whether dense retrieval methods maintain their advantages at industry scale remains an important open question and is beyond the scope of the current study.

6 Conclusion

In this work, we conducted a comprehensive comparison between dense retrieval methods and the emerging generative retrieval approach. Our analysis revealed the limitations of dense retrieval, including high computational and storage requirements, while highlighting the advantages of generative retrieval, which uses semantic IDs and generative models to enhance efficiency and semantic understanding. Furthermore, we have identified the challenges faced by generative retrieval, particularly in handling cold-start items and matching the performance of dense retrieval. To address these challenges, we introduced a novel hybrid model, LIGER, that combines the strengths of both approaches. Our findings demonstrate that our hybrid model surpasses existing models in handling cold-start scenarios and achieves advanced overall performance on benchmark datasets.

Looking ahead, the fusion of dense and generative retrieval methods holds tremendous potential for advancing recommendation systems. Our research provides a foundation for further exploration into hybrid models that capitalize on the strengths of both retrieval types. As these models continue to evolve, they will become increasingly practical for real-world applications, enabling more personalized and responsive user experiences.

References

Michele Bevilacqua, Giuseppe Ottaviano, Patrick Lewis, Wen tau Yih, Sebastian Riedel, and Fabio Petroni. Autoregressive search engines: Generating substrings as document identifiers. *ArXiv*, abs/2204.10628, 2022. URL https://api.semanticscholar.org/CorpusID:248366293.

Yuwei Cao, Nikhil Mehta, Xinyang Yi, Raghunandan Keshavan, Lukasz Heldt, Lichan Hong, Ed H Chi, and Maheswaran Sathiamoorthy. Aligning large language models with recommendation knowledge. *arXiv preprint arXiv:2404.00245*, 2024a.

Yuwei Cao, Nikhil Mehta, Xinyang Yi, Raghunandan H. Keshavan, Lukasz Heldt, Lichan Hong, Ed H. Chi, and Maheswaran Sathiamoorthy. Aligning large language models with recommendation knowledge. *ArXiv*, abs/2404.00245, 2024b. URL https://api.semanticscholar.org/CorpusID:268819967.

Jiangui Chen, Ruqing Zhang, J. Guo, M. de Rijke, Wei Chen, Yixing Fan, and Xueqi Cheng. Continual learning for generative retrieval over dynamic corpora. *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023. URL https://api.semanticscholar.org/CorpusID:261277063.

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. Autoregressive entity retrieval. *arXiv preprint arXiv:2010.00904*, 2020.

Gabriel de Souza Pereira Moreira, Sara Rabhi, Jeong Min Lee, Ronay Ak, and Even Oldridge. Transformers4rec: Bridging the gap between nlp and sequential/session-based recommendation. In *Proceedings of the 15th ACM conference on recommender systems*, pp. 143–153, 2021.

Yijie Ding, Yupeng Hou, Jiacheng Li, and Julian McAuley. Inductive generative recommendation via retrieval-based speculation. *ArXiv*, abs/2410.02939, 2024. URL https://api.semanticscholar.org/CorpusID:273163026.

Chao Feng, Wu Li, Defu Lian, Zheng Liu, and Enhong Chen. Recommender forest for efficient retrieval. In *Neural Information Processing Systems*, 2022. URL https://api.semanticscholar.org/CorpusID:258509354.

Jibril Frej, Marta Knezevic, and Tanja Käser. Graph reasoning for explainable cold start recommendation. *ArXiv*, abs/2406.07420, 2024. URL https://api.semanticscholar.org/CorpusID:270379698.

Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. *Proceedings of the 25th International Conference on World Wide Web*, 2016. URL https://api.semanticscholar.org/CorpusID:1964279.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997. URL https://api.semanticscholar.org/CorpusID:1915014.

Yupeng Hou, Zhankui He, Julian McAuley, and Wayne Xin Zhao. Learning vector-quantized item representation for transferable sequential recommenders. *Proceedings of the ACM Web Conference 2023*, 2022a. URL https://api.semanticscholar.org/CorpusID:253098091.

Yupeng Hou, Shanlei Mu, Wayne Xin Zhao, Yaliang Li, Bolin Ding, and Ji rong Wen. Towards universal sequence representation learning for recommender systems. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022b. URL https://api.semanticscholar.org/CorpusID:249625869.

Yupeng Hou, Shanlei Mu, Wayne Xin Zhao, Yaliang Li, Bolin Ding, and Ji-Rong Wen. Towards universal sequence representation learning for recommender systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 585–593, 2022c.

Feiran Huang, Zhen Yang, Junyi Jiang, Yuan-Qi Bei, Yijie Zhang, and Hao Chen. Large language model interaction simulator for cold-start item recommendation. *ArXiv*, abs/2402.09176, 2024. URL https://api.semanticscholar.org/CorpusID:267657482.

Hong Jun Jeon, Songbin Liu, Yuantong Li, Jie Lyu, Hunter Song, Ji Liu, Peng Wu, and Zheqing Zhu. Epinet for content cold start. *arXiv preprint arXiv:2412.04484*, 2024.

Mengqun Jin, Zexuan Qiu, Jieming Zhu, Zhenhua Dong, and Xiu Li. Contrastive quantization based semantic code for generative recommendation. *arXiv preprint arXiv:2404.14774*, 2024.

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*, pp. 197–206. IEEE, 2018a.

Wang-Cheng Kang and Julian J. McAuley. Self-attentive sequential recommendation. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*, pp. 197–206. IEEE Computer Society, 2018b. doi: 10.1109/ICDM.2018.00035.

Varsha Kishore, Chao gang Wan, Justin Lovelace, Yoav Artzi, and Kilian Q. Weinberger. Incdsi: Incrementally updatable document retrieval. *ArXiv*, abs/2307.10323, 2023. URL https://api.semanticscholar.org/CorpusID:259991824.

Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11523–11532, 2022a.

Hyunji Lee, Jaeyoung Kim, Hoyeon Chang, Hanseok Oh, Sohee Yang, Vladimir Karpukhin, Yi Lu, and Minjoon Seo. Contextualized generative retrieval. *ArXiv*, abs/2210.02068, 2022b. URL https://api.semanticscholar.org/CorpusID:252715634.

Hyunji Lee, Jaeyoung Kim, Hoyeon Chang, Hanseok Oh, Sohee Yang, Vladimir Karpukhin, Yi Lu, and Minjoon Seo. Nonparametric decoding for generative retrieval. In *Annual Meeting of the Association for Computational Linguistics*, 2022c. URL https://api.semanticscholar.org/CorpusID:258959550.

Jiacheng Li, Ming Wang, Jin Li, Jinmiao Fu, Xin Shen, Jingbo Shang, and Julian McAuley. Text is all you need: Learning language representations for sequential recommendation. *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023a. URL https://api.semanticscholar.org/CorpusID:258841284.

Jiacheng Li, Ming Wang, Jin Li, Jinmiao Fu, Xin Shen, Jingbo Shang, and Julian McAuley. Text is all you need: Learning language representations for sequential recommendation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1258–1267, 2023b.

Yongqi Li, Nan Yang, Liang Wang, Furu Wei, and Wenjie Li. Learning to rank in generative retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 8716–8723, 2024.

Han Liu, Yin wei Wei, Xuemeng Song, Weili Guan, Yuan-Fang Li, and Liqiang Nie. Mmgrec: Multimodal generative recommendation with transformer model. *ArXiv*, abs/2404.16555, 2024a. URL https://api.semanticscholar.org/CorpusID:269362930.

Yang Liu, Yitong Wang, and Chenyue Feng. Unirec: A dual enhancement of uniformity and frequency in sequential recommendations. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pp. 1483–1492, 2024b.

Sanket Vaibhav Mehta, Jai Gupta, Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Jinfeng Rao, Marc Najork, Emma Strubell, and Donald Metzler. Dsi++: Updating transformer memory with new documents. *ArXiv*, abs/2212.09744, 2022. URL https://api.semanticscholar.org/CorpusID:254854290.

Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith B Hall, Daniel Cer, and Yinfei Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. *arXiv preprint arXiv:2108.08877*, 2021.

Fabian Paischer, Liu Yang, Linfeng Liu, Shuai Shao, Kaveh Hassani, Jiacheng Li, Ricky Chen, Zhang Gabriel Li, Xialo Gao, Wei Shao, Xue Feng, Nima Noorshams, Sem Park, Bo Long, and Hamid Eghbalzadeh. Preference discerning with llm-enhanced generative retrieval. *ArXiv*, abs/2412.08604, 2024. URL https://api.semanticscholar.org/CorpusID:274638830.

Haohao Qu, Wenqi Fan, Zihuai Zhao, and Qing Li. Tokenrec: Learning to tokenize id for llm-based generative recommendation. *ArXiv*, abs/2406.10450, 2024. URL https://api.semanticscholar.org/CorpusID:270560438.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. URL https://api.semanticscholar.org/CorpusID:160025533.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Tran, Jonah Samost, et al. Recommender systems with generative retrieval. *Advances in Neural Information Processing Systems*, 36, 2024.

Kaushik Rangadurai, Yiqun Liu, Siddarth Malreddy, Xiaoyi Liu, Piyush Maheshwari, Vishwanath Sangale, and Fedor Borisyuk. Nxtpost: User to post recommendations in facebook groups. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3792–3800, 2022.

Scott Sanner, Krisztian Balog, Filip Radlinski, Benjamin D. Wedin, and Lucas Dixon. Large language models are competitive near cold-start recommenders for language- and item-based preferences. *Proceedings of the 17th ACM Conference on Recommender Systems*, 2023. URL https://api.semanticscholar.org/CorpusID:260164942.

Anima Singh, Trung Vu, Nikhil Mehta, Raghunandan Keshavan, Maheswaran Sathiamoorthy, Yilin Zheng, Lichan Hong, Lukasz Heldt, Li Wei, Devansh Tandon, et al. Better generalization with semantic ids: A case study in ranking for recommendations. *arXiv preprint arXiv:2306.08121*, 2023.

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 1441–1450, 2019.

Weiwei Sun, Lingyong Yan, Zheng Chen, Shuaiqiang Wang, Haichao Zhu, Pengjie Ren, Zhumin Chen, Dawei Yin, Maarten Rijke, and Zhaochun Ren. Learning to tokenize for generative retrieval. *Advances in Neural Information Processing Systems*, 36, 2024.

Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. Transformer memory as a differentiable search index. *ArXiv*, abs/2202.06991, 2022. URL https://api.semanticscholar.org/CorpusID:246863488.

Hai Dang Tran and Andrew Yates. Dense retrieval with entity views. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pp. 1955–1964, 2022.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017. URL https://api.semanticscholar.org/CorpusID:13756489.

Jianling Wang, Haokai Lu, James Caverlee, Ed H Chi, and Minmin Chen. Large language models as data augmenters for cold-start item recommendation. In *Companion Proceedings of the ACM Web Conference 2024*, pp. 726–729, 2024.

Yujing Wang, Ying Hou, Hong Wang, Ziming Miao, Shibin Wu, Hao Sun, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, Xing Xie, Hao Sun, Weiwei Deng, Qi Zhang, and Mao Yang. A neural corpus indexer for document retrieval. *ArXiv*, abs/2206.02743, 2022. URL `https://api.semanticscholar.org/CorpusID:249395549`.

D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. doi: 10.1109/4235.585893.

Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed H. Chi. Sampling-bias-corrected neural modeling for large corpus item recommendations. *Proceedings of the 13th ACM Conference on Recommender Systems*, 2019. URL `https://api.semanticscholar.org/CorpusID:202639719`.

Tingting Zhang, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Deqing Wang, Guanfeng Liu, Xiaofang Zhou, et al. Feature-level deeper self-attention network for sequential recommendation. In *IJCAI*, pp. 4320–4326, 2019.

Yin Zhang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Lichan Hong, and Ed H. Chi. A model of two tales: Dual transfer learning framework for improved long-tail item recommendation. *Proceedings of the Web Conference 2021*, 2020. URL `https://api.semanticscholar.org/CorpusID:226221746`.

Yin Zhang, Ruoxi Wang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Lichan Hong, James Caverlee, and Ed H. Chi. Empowering long-tail item recommendation through cross decoupling network (cdn). *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022. URL `https://api.semanticscholar.org/CorpusID:253117145`.

Bowen Zheng, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, and Ji rong Wen. Adapting large language models by integrating collaborative semantics for recommendation. *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pp. 1435–1448, 2023. URL `https://api.semanticscholar.org/CorpusID:265213194`.

Bowen Zheng, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, Ming Chen, and Ji-Rong Wen. Adapting large language models by integrating collaborative semantics for recommendation. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pp. 1435–1448. IEEE, 2024.

Carolina Zheng, Minhui Huang, Dmitrii Pedchenko, Kaushik Rangadurai, Siyu Wang, Gaby Nahum, Jie Lei, Yang Yang, Tao Liu, Zutian Luo, et al. Enhancing embedding representation stability in recommendation systems with semantic id. *arXiv preprint arXiv:2504.02137*, 2025.

Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pp. 1893–1902, 2020.

A Related Work

Generative Retrieval. The concept of generative retrieval was first proposed by Tay et al. (2022) within the domain of document retrieval. This paradigm shifts from traditional search and retrieval methods by encoding document information directly into the weights of a Transformer model. Subsequent studies (De Cao et al., 2020; Bevilacqua et al., 2022; Feng et al., 2022) have expanded on this foundation, enhancing document retrieval through improvements in indexing (Lee et al., 2022b;c; Wang et al., 2022), and the efficient continual database updates (Mehta et al., 2022; Kishore et al., 2023; Chen et al., 2023).

In the realm of sequential recommendation systems, Rajput et al. (2024) is the first work to leverage the generative retrieval techniques. The target item is directly generated given a user's interaction history, rather than selecting top items by ranking all relevant user-item pairs. A key challenge in generative retrieval is striking a balance between memorization and generalization when encoding items. To address this, semantic

IDs have been proposed by leveraging RQ-VAE models (Lee et al., 2022a; Van Den Oord et al., 2017). These models encode content-based embeddings into a compact, discrete semantic indexer that captures the hierarchical structure of concepts within an item's content, proving to be scalable in industrial applications (Singh et al., 2023). Recent developments (Hou et al., 2022a) have expanded semantic-ID-based generative retrieval to include contrastive learning (Jin et al., 2024), multimodal integration (Liu et al., 2024a), tokenization techniques (Sun et al., 2024), learning-to-rank methods (Li et al., 2024), and improved RQ-VAE training (Qu et al., 2024).

Sequential Dense Recommendation. Traditional sequential dense recommender models follow the paradigm of learning representations of users, items, and their interactions with multimodal data. Early work (Hidasi et al., 2015) proposed architectures based on traditional Recurrent Neural Networks (RNNs), while later studies (Kang & McAuley, 2018a; Sun et al., 2019; de Souza Pereira Moreira et al., 2021) have shifted towards the Transformer architecture to enhance performance. Besides capturing the user-item interaction history pattern with the sequential modeling, extra features such as item attributes (Zhang et al., 2019; Zhou et al., 2020) has been utilized to further improve the performance. With the recent advancements in Large Language Models (LLMs), several works have explored using these models as the backbone for recommender systems, aligning item representations with LLMs to improve recommendation performance (Li et al., 2023b; Hou et al., 2022c; Cao et al., 2024a; Zheng et al., 2024). In this work, we aim to merge the sequential dense recommendation approach with generative retrieval techniques, assessing performance gaps and resource efficiency, and proposing a hybrid method that combines the strengths of both paradigms.

Cold-start Problem. Traditional challenges such as long-tail and cold-start items continue to hinder recommendation systems. The long-tail items issue arises from skewed distributions where a few popular items dominate user interactions (Zhang et al., 2022; 2020), while the cold-start problem arises when new items are introduced without any historical interaction data. This problem is particularly prevalent in industrial settings (Rangadurai et al., 2022; Jeon et al., 2024), where systems must frequently incorporate new items under tight latency and scale constraints. Recent studies (Hou et al., 2022c; Li et al., 2023b) have shown that textual embeddings can provide a robust prior for tackling the cold-start issue, and further improvements have been achieved by integrating pretrained LLMs (Huang et al., 2024; Sanner et al., 2023; Wang et al., 2024) and knowledge graphs (Frej et al., 2024). In this work, we explore the cold-start problem within the context of generative retrieval and propose a hybrid method that combines dense retrieval with textual embeddings to effectively mitigate this issue. A concurrent work by Ding et al. (2024) also investigates the cold-start issue in generative retrieval and proposes using dense retrieval as a drafter model to facilitate cold-start item generation.

B Experimental Details

B.1 Implementation Details of Dense Retrieval

In Section 4.1, we provide the TIGER implementation details. Here, we describe the implementation details of the dense retrieval method we developed.

The dense retrieval method uses the same T5-encoder architecture, configured with 6 layers, an embedding dimension of 128, 6 attention heads, and a feed-forward network hidden dimension of 1024. For content embeddings, we use the same model as TIGER, sentence-T5-XXL, which generates text embeddings in $\mathbf{R}^{768}$. To integrate these embeddings into the Transformer, we project them down to a 128-dimensional space using a `Linear` layer.

Following the notation in Section 2.2, once $\mathbf{E}_i$ is collected for each item, it is passed through a `LayerNorm` layer, followed by a `Dropout` layer with a rate of 0.5. For the cosine similarity loss calculation, we set the temperature parameter $\tau = 0.07$. During training, we use the same learning rate (0.0003), cosine learning rate scheduler, optimizer (AdamW), and weight decay (0.035) as employed in TIGER's training.

Table 3: Dataset statistics after applying 5-core filtering to both users and items. The first three datasets (Beauty, Sports, and Toys) are subsets of the Amazon review dataset.

Dataset	# users	# items	# actions	# cold-start items
Beauty	22,363	12,101	198,502	43
Toys and Games	19,412	11,924	167,597	56
Sports and Outdoors	35,598	18,357	296,337	81
Steam	47,761	12,012	599,620	400

B.2 Implementation Difference Between UniSRec and our Dense Model

Although the dense retrieval model we implemented is inspired by UniSRec (Hou et al., 2022c), we made several modifications to simplify the model and align it with the best-tuned TIGER architecture. Specifically:

1. We use the same content model as TIGER: T5-XXL, whereas UniSRec uses BERT.
2. We adopt the same encoder architecture as TIGER, which consists of 6 T5 encoder layers with 6 attention heads, an embedding dimension of 128, and a feed-forward network hidden dimension of 1024. In contrast, UniSRec uses a custom Transformer block with 2 layers, 2 attention heads, an embedding size of 300, a hidden size of 256, and different implementations for `LayerNorm` and positional embeddings compared to the T5 model [3].
3. We replace UniSRec's mixture-of-expert layer and whitening layer with a simple `Linear` layer.
4. We train the dense retrieval model from scratch, whereas UniSRec relies on pretraining using the Amazon 2018 datasets.

B.3 Data Statistics

In Table 3, we present the statistics of the datasets used in our evaluation.

B.4 Baselines

We compare our methods with five state-of-the-art Item-ID-based dense retrieval methods, including:

1. SASRec (Kang & McAuley, 2018b). A self-attention based sequential recommendation model that learns to predict the next item ID based on the user's interaction history.
2. FDSA (Zhang et al., 2019) [*feature-informed*]. This method extends SASRec by incorporating item features into the self-attention model, allowing it to leverage prior information about cold-start items through their attributes.
3. S^3-Rec (Zhou et al., 2020) [*feature-informed*]. A self-attention based model that utilizes data correlation to create self-supervision signals, improving sequential recommendation through pre-training.
4. UnisRec (Hou et al., 2022b) [*modality-based*]. A model that learns universal item representations by utilizing associated description text and a lightweight encoding architecture that incorporates parametric whitening and a mixture-of-experts adaptor. We fine-tune the released pretrained model in the transductive setting.
5. Recformer (Li et al., 2023a) [*modality-based*]. A bidirectional Transformer-based model that encodes item information using key-value attributes described by text. We fine-tune the pre-trained model on the downstream datasets.

[3] `https://github.com/RUCAIBox/UniSRec/blob/master/props/UniSRec.yaml`

C Full Experimental Result

In Table 4, we present the full results on the benchmark, where our method with different number of retrieved candidates from generative retrieval are shown.

D Ablation Study

LIGER combines dense retrieval and generative retrieval paradigms into a unified framework. As described in Section 3 and illustrated in Figure 2, for each item i with semantic ID $(s_i^1, s_i^2, \cdots, s_i^m)$, we construct the input embedding for each semantic ID as:

$$\mathbf{E}_{s_i^j} = \mathbf{e}_{s_i^j} + \mathbf{e}_i^{\texttt{text}} + \mathbf{e}_i^{\texttt{pos}} + \mathbf{e}_j^{\texttt{pos}},$$

where $\mathbf{e}_{s_i^j}$ is the learnable embedding for the s_i^j semantic ID, $\mathbf{e}_i^{\texttt{text}}$ is the item's text representation, $\mathbf{e}_i^{\texttt{pos}}$ is the positional embedding for the item, and $\mathbf{e}_j^{\texttt{pos}}$ is the positional embedding for the semantic ID. The final embedding for item i is then represented as: $\mathbf{E}_i = [\mathbf{E}_{s_i^1}, \mathbf{E}_{s_i^2}, \cdots, \mathbf{E}_{s_i^m}]$.

During training, the model is optimized with two objectives: the cosine similarity loss and the next-token prediction loss on the semantic IDs of the next item: The cosine similarity loss ensures that the model learns to align the *encoder*'s output embedding with the text representation of the next item, and the next-token prediction loss supervise on the next item's semantic ID tuple prediction performance. To summarize, the LIGER framework is structured as follows:

1. *Input:* Semantic ID (SID) and item's text representation;
2. *Output*: Predictions for:
   (a) The next item's semantic ID through the SID head.
   (b) The next item's text representation through the embedding head.

The ablation study investigates the impact of each component, as shown in Figure 7:

- *LIGER (detach)* detaches the gradient updates from the SID head in LIGER to examine the importance of multi-objective optimization through the SID head;
- *TIGER (T)* removes the embedding head from LIGER, focusing solely on the SID head and the text representation as input;
- *TIGER* further simplifies *TIGER (T)* by removing the item text representation input, reducing the model to the generative retrieval method described in Section 2.1;
- *Dense (SID)* removes the SID head from LIGER, retaining only the dense retrieval mechanism with SID as input;
- *Dense* replaces the SID input with ID in *Dense (SID)*, reducing the model to the dense retrieval method in transductive setting, as detailed in Section 2.2.

Figure 8 presents the ablation results in terms of Recall@10 across four datasets: Beauty, Sports, Toys, and Steam. The performance is analyzed with respect to the number of candidates retrieved by the SID head (K).

First, comparing *TIGER* to *TIGER(T)*, we observe that *TIGER(T)* consistently performs the same or slightly better, demonstrating the positive impact of incorporating the item's text representation as input. However, the improvement is modest, indicating that while text representation is helpful, its contribution alone does not significantly enhance the model's performance.

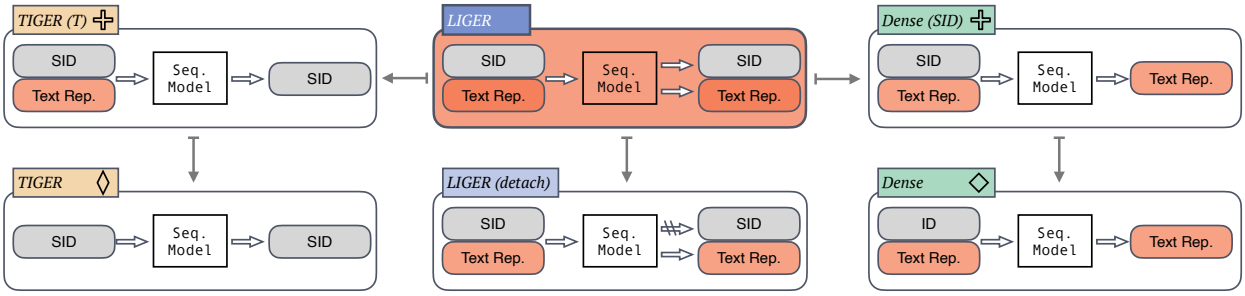


Figure 7: *Overview of our Ablation Study.* This study examines the effects of different components within LIGER (*top middle*), which integrates TIGER and semantic ID (SID)-based dense retrieval in an transductive setting. LIGER takes both the semantic ID and item text representation as inputs, predicting the SID and generating embeddings. We perform the following ablations to evaluate the impact of specific components: (1) To assess the effect of multi-objective optimization, we detach the gradient updates from the SID head (*bottom middle*). (2) To study the role of the embedding head, we remove it (*top left*). (3) To evaluate the contribution of the item text representation input in (2), we remove it, reducing the model to TIGER (*bottom left*). (4) To analyze the effect of the SID head, we remove it (*top right*). (5) Finally, we replace the SID with item IDs in (4), reducing the model to standard dense retrieval in transductive setting (*bottom right*).

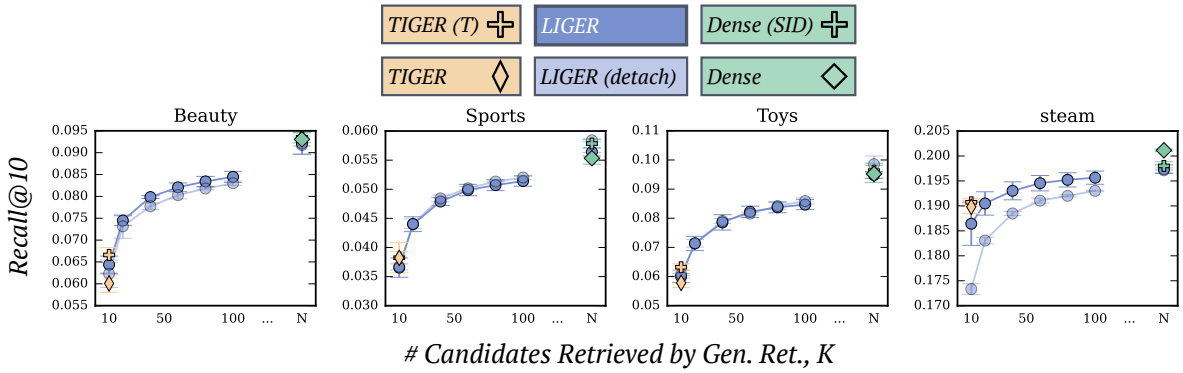


Figure 8: *Ablation Results on Recall@10 across Datasets.*

Second, when comparing *Dense* retrieval with *Dense (SID)*, the results are similar, suggesting that the primary limitation of the dense retrieval method is not due to the type of representation used (ID vs. SID). This highlights that the bottleneck contributing to the performance difference lies elsewhere, possibly in the learning of SID.

LIGER, which combines *TIGER* and *dense* retrieval paradigms, exhibits a smooth interpolation between the performance of *TIGER* and *Dense*. This suggests that LIGER effectively leverages the strengths of both approaches to achieve robust performance across datasets. Notably, when the gradient update from the SID head is detached (*LIGER(detach)*), the model still performs comparably to the standard LIGER, with the most significant drop observed on the Steam dataset. This result implies that the SID head's learning signal is crucial for the Steam dataset, which is of a larger scale compared to the Amazon datasets.

Overall, these results demonstrate that LIGER strikes a balance between TIGER and dense retrieval methods while retaining flexibility through multi-objective optimization. However, the importance of the SID head's gradient signals appears dataset-dependent, possibly also influenced by dataset scales, as highlighted by the performance gap on Steam.

Table 4: *Performance Table for Amazon Beauty, Sports, Toys, and Steam Datasets Across Various Baseline Methods.* In this table, we present our method with different number of retrieved candidates K from generative retrieval.

Datasets	Methods	Inference Cost	**NDCG@10**↑ (%)		**Recall@10**↑ (%)	
			In-set	Cold	In-set	Cold
Beauty	SASRec	$\mathcal{O}(N)$	2.179 ± 0.023	0.000 ± 0.000	5.109 ± 0.042	0.000 ± 0.000
	FDSA	$\mathcal{O}(N)$	2.244 ± 0.135	0.000 ± 0.000	4.530 ± 0.357	0.000 ± 0.000
	S^3-Rec	$\mathcal{O}(N)$	2.279 ± 0.058	0.000 ± 0.000	5.226 ± 0.229	0.000 ± 0.000
	UniSRec	$\mathcal{O}(N)$	3.346 ± 0.057	1.422 ± 0.128	6.937 ± 0.110	3.704 ± 0.000
	RecFormer	$\mathcal{O}(N)$	2.880 ± 0.085	1.955 ± 0.433	6.265 ± 0.196	4.733 ± 0.943
	TIGER	$\mathcal{O}(tK)$	3.216 ± 0.084	0.000 ± 0.000	6.009 ± 0.204	0.000 ± 0.000
	Ours ($K = 20$)	$\mathcal{O}(tK)$	4.020 ± 0.044	3.800 ± 0.523	7.447 ± 0.111	10.082 ± 1.285
	Ours ($K = 40$)	$\mathcal{O}(tK)$	4.230 ± 0.067	2.815 ± 0.525	7.985 ± 0.033	7.613 ± 1.285
	Ours ($K = 60$)	$\mathcal{O}(tK)$	4.328 ± 0.113	2.609 ± 0.343	8.207 ± 0.101	7.202 ± 0.943
	Ours ($K = 80$)	$\mathcal{O}(tK)$	4.392 ± 0.097	2.550 ± 0.425	8.343 ± 0.115	7.202 ± 0.943
	Ours ($K = 100$)	$\mathcal{O}(tK)$	4.430 ± 0.101	2.469 ± 0.381	8.447 ± 0.126	7.202 ± 0.943
	Ours ($K = N$)	$\mathcal{O}(N)$	4.738 ± 0.151	1.225 ± 0.256	9.210 ± 0.247	3.704 ± 0.617
Sports	SASRec	$\mathcal{O}(N)$	1.160 ± 0.038	0.000 ± 0.000	2.696 ± 0.102	0.000 ± 0.000
	FDSA	$\mathcal{O}(N)$	1.391 ± 0.162	0.000 ± 0.000	2.699 ± 0.312	0.000 ± 0.000
	S^3-Rec	$\mathcal{O}(N)$	1.097 ± 0.033	0.000 ± 0.000	2.557 ± 0.034	0.000 ± 0.000
	UniSRec	$\mathcal{O}(N)$	1.814 ± 0.041	0.676 ± 0.244	3.753 ± 0.106	1.559 ± 0.447
	RecFormer	$\mathcal{O}(N)$	1.318 ± 0.053	1.797 ± 0.000	2.921 ± 0.167	3.801 ± 0.000
	TIGER	$\mathcal{O}(tK)$	1.989 ± 0.085	0.064 ± 0.056	3.822 ± 0.109	0.195 ± 0.169
	Ours ($K = 20$)	$\mathcal{O}(tK)$	2.430 ± 0.075	2.731 ± 0.229	4.400 ± 0.127	5.848 ± 0.292
	Ours ($K = 40$)	$\mathcal{O}(tK)$	2.594 ± 0.063	1.814 ± 0.052	4.789 ± 0.067	4.191 ± 0.338
	Ours ($K = 60$)	$\mathcal{O}(tK)$	2.672 ± 0.067	1.517 ± 0.068	4.987 ± 0.098	3.411 ± 0.338
	Ours ($K = 80$)	$\mathcal{O}(tK)$	2.714 ± 0.064	1.270 ± 0.096	5.071 ± 0.094	2.924 ± 0.506
	Ours ($K = 100$)	$\mathcal{O}(tK)$	2.744 ± 0.055	1.175 ± 0.059	5.141 ± 0.087	2.729 ± 0.169
	Ours ($K = N$)	$\mathcal{O}(N)$	2.962 ± 0.053	0.581 ± 0.238	5.641 ± 0.071	1.365 ± 0.447
Toys	SASRec	$\mathcal{O}(N)$	2.756 ± 0.079	0.000 ± 0.000	6.314 ± 0.178	0.000 ± 0.000
	FDSA	$\mathcal{O}(N)$	2.375 ± 0.277	0.000 ± 0.000	4.684 ± 0.483	0.000 ± 0.000
	S^3-Rec	$\mathcal{O}(N)$	2.942 ± 0.071	0.000 ± 0.000	6.659 ± 0.135	0.000 ± 0.000
	UniSRec	$\mathcal{O}(N)$	3.622 ± 0.056	1.090 ± 0.084	7.472 ± 0.058	2.477 ± 0.195
	RecFormer	$\mathcal{O}(N)$	3.697 ± 0.052	4.432 ± 0.094	7.971 ± 0.170	10.023 ± 0.516
	TIGER	$\mathcal{O}(tK)$	2.949 ± 0.049	0.000 ± 0.000	5.782 ± 0.163	0.000 ± 0.000
	Ours ($K = 20$)	$\mathcal{O}(tK)$	3.756 ± 0.151	5.231 ± 0.531	7.135 ± 0.244	13.063 ± 0.516
	Ours ($K = 40$)	$\mathcal{O}(tK)$	4.021 ± 0.157	3.574 ± 0.262	7.859 ± 0.264	9.459 ± 0.585
	Ours ($K = 60$)	$\mathcal{O}(tK)$	4.163 ± 0.140	3.173 ± 0.149	8.222 ± 0.194	8.559 ± 0.516
	Ours ($K = 80$)	$\mathcal{O}(tK)$	4.245 ± 0.130	2.861 ± 0.139	8.375 ± 0.187	7.770 ± 0.338
	Ours ($K = 100$)	$\mathcal{O}(tK)$	4.288 ± 0.120	2.783 ± 0.108	8.468 ± 0.172	7.658 ± 0.195
	Ours ($K = N$)	$\mathcal{O}(tK)$	4.680 ± 0.086	2.149 ± 0.202	9.482 ± 0.117	6.081 ± 0.676
Steam	SASRec	$\mathcal{O}(N)$	14.763 ± 0.051	0.000 ± 0.000	18.259 ± 0.055	0.000 ± 0.000
	FDSA	$\mathcal{O}(N)$	8.236 ± 0.152	0.000 ± 0.000	14.773 ± 0.234	0.000 ± 0.000
	S^3-Rec	$\mathcal{O}(N)$	14.437 ± 0.127	0.000 ± 0.000	18.025 ± 0.222	0.000 ± 0.000
	UniSRec	$\mathcal{O}(N)$	-	-	-	-
	RecFormer	$\mathcal{O}(N)$	14.034 ± 0.123	0.120 ± 0.037	17.042 ± 0.275	0.319 ± 0.110
	TIGER	$\mathcal{O}(tK)$	15.034 ± 0.064	0.000 ± 0.000	18.980 ± 0.135	0.000 ± 0.000
	Ours ($K = 20$)	$\mathcal{O}(tK)$	14.951 ± 0.158	0.512 ± 0.047	19.049 ± 0.234	1.466 ± 0.110
	Ours ($K = 40$)	$\mathcal{O}(tK)$	15.138 ± 0.110	0.298 ± 0.068	19.302 ± 0.180	0.829 ± 0.110
	Ours ($K = 60$)	$\mathcal{O}(tK)$	15.236 ± 0.074	0.258 ± 0.109	19.455 ± 0.154	0.701 ± 0.292
	Ours ($K = 80$)	$\mathcal{O}(tK)$	15.284 ± 0.059	0.226 ± 0.105	19.522 ± 0.143	0.637 ± 0.292
	Ours ($K = 100$)	$\mathcal{O}(tK)$	15.318 ± 0.049	0.222 ± 0.109	19.566 ± 0.132	0.637 ± 0.292
	Ours ($K = N$)	$\mathcal{O}(tK)$	15.431 ± 0.006	0.175 ± 0.082	19.736 ± 0.086	0.510 ± 0.221