Recommender Systems with Generative Retrieval

Shashank Rajput[*1], Nikhil Mehta[*2], Anima Singh[2], Raghunandan Keshavan[2], Trung Vu[2], Lukasz Heldt[2], Lichan Hong[2], Yi Tay[2], Vinh Q. Tran[2], Jonah Samost[2], Maciej Kula[2], Ed H. Chi[2], Maheswaran Sathiamoorthy[2]

[1]University of Wisconsin-Madison, [2]Google

United States

ABSTRACT

Modern recommender systems leverage large-scale retrieval models consisting of two stages: training a dual-encoder model to embed queries and candidates in the same space, followed by an Approximate Nearest Neighbor (ANN) search to select top candidates given a query's embedding. In this paper, we propose a new single-stage paradigm: a generative retrieval model which autoregressively decodes the identifiers for the target candidates in one phase. To do this, instead of assigning randomly generated atomic IDs to each item, we generate Semantic IDs: a semantically meaningful tuple of codewords for each item that serves as its unique identifier. We use a hierarchical method called RQ-VAE to generate these codewords. Once we have the Semantic IDs for all the items, a Transformer based sequence-to-sequence model is trained to predict the Semantic ID of the next item. Since this model predicts the tuple of codewords identifying the next item directly in an autoregressive manner, it can be considered a generative retrieval model. We show that our recommender system trained in this new paradigm improves the results achieved by current SOTA models on the Amazon dataset. Moreover, we demonstrate that the sequence-to-sequence model coupled with hierarchical Semantic IDs offers better generalization and hence improves retrieval of cold-start items for recommendations.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

Recommender Systems, Generative Retrieval, Vector Quantization

ACM Reference Format:

Shashank Rajput[*1], Nikhil Mehta[*2], Anima Singh[2], Raghunandan Keshavan[2], Trung Vu[2], Lukasz Heldt[2], Lichan Hong[2], Yi Tay[2], Vinh Q. Tran[2], Jonah Samost[2], Maciej Kula[2], Ed H. Chi[2], Maheswaran Sathiamoorthy[2]. 2018. Recommender Systems with Generative Retrieval. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

[*] denotes Equal contribution.

Correspondence to rajput3@wisc.edu, nikhilmehta@google.com, nlogn@google.com.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00

https://doi.org/XXXXXXX.XXXXXXX

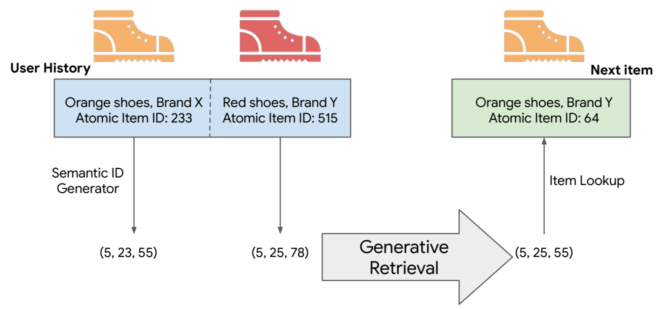


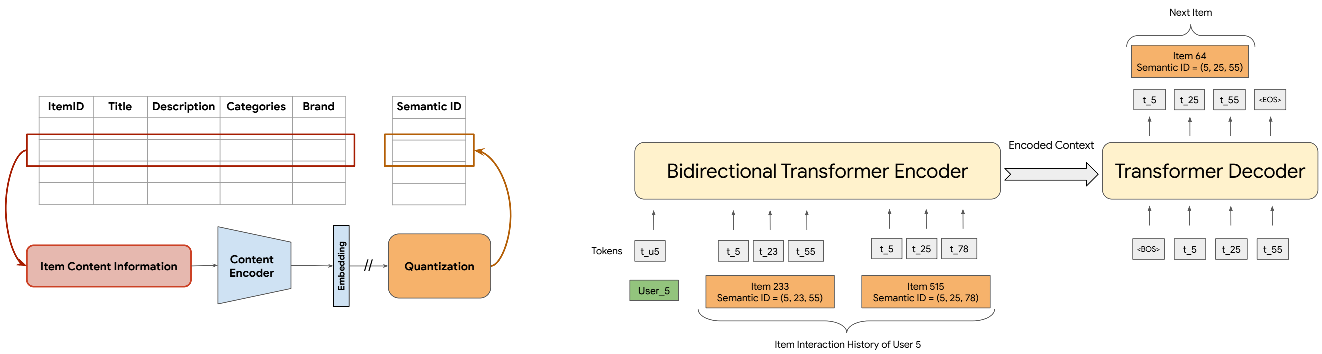
Figure 1: High-level overview of *Transformer Index for GEnerative Recommenders* (TIGER) framework. TIGER proposes representing an item as a tuple of discrete semantic tokens (referred to as Semantic ID), which allows framing the sequential recommendation task as a generative task such that the Semantic ID of the next item is directly predicted using a sequence-to-sequence model.

1 INTRODUCTION

Recommender systems help users discover content of interest and are ubiquitous in various recommendation domains such as video [3, 9, 45], app [2], product [6, 8], and music [18, 19]. Modern recommender systems adopt a retrieve-and-rank strategy, where a set of viable candidates are selected in the retrieval stage, which are then ranked using a ranker model. Since the ranker model works only on the candidates it receives, it is desired that the retrieval stage emits highly relevant candidates.

There are standard and well-established methods for building retrieval models. Matrix factorization [19] learns query and candidate embeddings in the same space. In order to better capture the non-linearities in the data, dual-encoder architectures [39] (i.e., one tower for the query and another for the candidate) employing inner-product to embed the query and candidate embeddings in the same space have become popular in recent years. To use these models during inference, an index that stores the embeddings for all items is created using the candidate tower. For a given query, its embedding is calculated using the query tower, and then an Approximate Nearest Neighbors (ANN) algorithm is used to choose the nearest candidates. On the other hand, sequential recommenders [6, 11, 17, 23, 30, 43, 46], which explicitly take into account the order of user-item interactions, are also recently popular. They also typically use softmax in the output layer when training and resort to using ANN during inference.

We propose a new paradigm of building generative retrieval models for sequential recommenders. Instead of traditional query-candidate matching approaches, our method is uses an end-to-end



(a) Semantic ID generation for items using quantization over content embeddings derived from content information.

(b) The proposed Transformer based Encoder-Decoder setup for building the sequence-to-sequence model used for generative retrieval.

Figure 2: A high-level overview of the modeling approach used in TIGER.

generative model that predicts the candidate IDs directly - dispensing the need for any discrete, non-differentiable inner-product search system or index altogether. Using autoregressive decoding [26, 27] and beam search [31, 38], we can retrieve several viable candidates. In this case, we can interpret the Transformer [35] memory (parameters) as an end-to-end recommendation index, reminiscent of Tay et al. [33]. As such, we name our proposed method TIGER, short for *Transformer Index for GEnerative Recommenders*. A high-level overview of TIGER is shown in Figure 1.

Critically, TIGER is characterized by a new approach to represent each item by a novel "Semantic ID": a sequence of tokens based on the content information about the item (such as its text description). Concretely, given an item's text description, we can use pre-trained text encoders (e.g., SentenceT5 [25]) to generate dense content embeddings. A quantization scheme can then be applied over the embeddings to form a small set of tokens/codewords (integers). We refer to this ordered tuple of codewords as the Semantic ID of the item. We derive inspiration for this idea from human language, where we have words for concepts and string together words to convey complex ideas. Similarly, we want to develop a language for IDs to represent items. This way, we can represent billions of items using a sequence of limited sets of words (tokens), in contrast to randomly generated atomic IDs requiring one ID per item.

Notably, similar ideas have been adopted when generating images from text (e.g., Parti [41]), where images are represented as tokens using ViT-VQGAN [40]. One crucial difference is that in image generation, a few incorrect tokens lead to a small error or noise in the image. In contrast, a single incorrect token here would mean that the recommender system predicts a different or non-existent item.

There are many benefits to using these Semantic IDs. The number of users and items in modern recommender systems can be in the billions. Thus the embedding tables of ANN-based models can become prohibitively large when using 1:1 mapping between atomic IDs and embedding vectors. Not only does this require a huge memory and storage footprint, but it also leads to imbalance when training the embeddings of the items, where popular items will be over-sampled as compared to infrequent ones [39]. As such, it is common practice to maintain dedicated vocabulary for more popular items and randomly hash remaining items in a fixed set of buckets[16]. However, the hashing scheme leads to random collisions. As new items are introduced in real-world recommendation products, the random collisions could aggravate the cold-start problem. In contrast, our method requires maintaining an embedding table for only a small set of tokens. Furthermore, the collisions in our case are semantically meaningful, which helps mitigate the cold-start problem.

We summarize the main contributions of this work below:

(1) We propose TIGER, a novel generative retrieval-based recommender model, which first assigns unique Semantic IDs to each item, and then trains a retrieval model to predict the Semantic ID of the next item the user will engage with. This provides an alternative to the high-dimensional nearest neighbor search-based or softmax-based recommender systems.
(2) We show that TIGER outperforms existing SOTA recommender systems' recall and NDCG metrics on multiple datasets.
(3) We find that this new paradigm of generative retrieval leads to two extra capabilities in sequential recommender systems: 1. Being able to recommend new and infrequent items, thus improving cold-start problems, and 2. Being able to generate diverse recommendations using a tunable parameter.

Paper Overview. In Section 2, we provide a brief literature survey of the techniques used in recommender systems, generative retrieval, and the particular Semantic ID generation techniques we use in this paper. In Section 3, we explain our proposed framework, outline the various techniques we use for Semantic ID generation, and provide the results of our experiments in Section 4. We provide further discussions about our work along with some insights in Section 5 and conclude the paper in Section 6.

2 RELATED WORK

Sequential Recommenders. Using deep sequential models in recommender systems has developed into a rich literature. GRU4REC [11] was the first to use GRU based RNNs for sequential recommendations. Li *et al.* [23] proposed NARM (Neural Attentive Session-based Recommendation), where they used the attention mechanism along with a GRU layer to track both the long term and the current intent of the user. AttRec [43] proposed by Zhang *et al.* used the self-attention mechanism to model the user's intent in the current session, and added personalization by modeling user-item affinity separately using metric learning. Concurrently, Kang *et al.* proposed SASRec [17], which used self-attention similar to decoder-only transformer models.

Inspired by the success of masked language modeling in language tasks, BERT4Rec [30] and Transformers4Rec [6] utilize transformer models with masking strategies for sequential recommendation tasks. S^3-Rec [46] goes beyond just masking by pre-training on four self-supervised tasks to improve data representation. In a concurrent work to us, Hou *et al.* propose VQ-Rec [12], which generates "codes" (analogous to Semantic IDs that we use) using content information to represent items. Their focus is on building transferable recommender systems, and do not use the codes in a generative manner for retrieval. While they use product quantization [15] to generate the codes, we use RQ-VAE to generate the Semantic ID.

All of the models described above learn a separate high-dimensional embedding for each item and then perform an ANN or Maximum Inner Product Search (MIPS) to predict the next item. In contrast, our proposed technique, TIGER, uses Generative Retrieval to directly predict the Semantic ID of the next item. P5 [8] and M6 [4] fine-tune pre-trained large language models, to get *multi-task* recommender systems that output token-by-token the recommended item. For P5, the model relies on the tokenizer used by the LLM (SentencePiece tokenizer [29]) to generate tokens out of item IDs that are non-semantic in nature. M6, on the other hand, directly outputs the name of the recommended item, token-by-token. We use a more principled way of generating Semantic IDs, based on content information, that is compatible with most existing sequence-to-sequence models and show in Table 3 that our proposed approach yields much better results than using tuples of random codes.

Generative Retrieval. Generative retrieval is a recently developed approach for Document Retrieval in the NLP community, where the task is to return a set of relevant documents from a database. Document retrieval traditionally involved training a 2-tower model which mapped both queries and documents to the same high-dimensional vector space, followed by performing an ANN or MIPS for the query over all the documents to return the closest ones. This technique presents some disadvantages like having a large embedding table [21, 22]. Generative retrieval is a recently proposed technique that aims to fix some of the issues of the traditional approach by producing token by token either the title, name, or the document id string of the document. Cao *et al.* [5] proposed GENRE for entity retrieval, which used a transformer-based architecture to return, token-by-token, the name of the entity referenced to in a given query. Tay *et al.* [33] proposed DSI for document retrieval, which was the first system to assign structured semantic DocIDs to each document. Then given a query, the model autoregressively returned the DocID of the document token-by-token. The DSI work marks a paradigm shift in IR to generative retrieval approaches and is the first successful application of an end-to-end Transformer for retrieval applications. Subsequently, Lee *et al.* [22] show that generative document retrieval is useful even in the multi-hop setting, where a complex query cannot be answered directly by a single document, and hence their model generates intermediate queries, in a chain-of-thought manner, to ultimately generate the output for the complex query. Wang *et al.* [36] supplement the hierarchical k-means clustering based semantic DocIDs of Tay *et al.* [33] by proposing a new decoder architecture that specifically takes into account the prefixes in semantic DocIDs. In CGR [21], the authors propose a way to take advantage of both the bi-encoder technique and the generative retrieval technique, by allowing the decoder, of their encoder-decoder-based model, to learn separate *contextualized* embeddings which store information about documents intrinsically. To the best of our knowledge, we are the first to use generative Semantic IDs created using an auto-encoder (RQ-VAE [20, 42]) for retrieval models.

Vector Quantization. We refer to Vector Quantization as the process of converting a high-dimensional vector into a low-dimensional tuple of codewords. One of the most straightforward techniques uses hierarchical clustering, such as the one used in [33], where clusters created in a particular iteration are further partitioned into sub-clusters in the next iteration. An alternative popular approach is Vector-Quantized Variational AutoEncoder (VQ-VAE), which was introduced in [34] as a way to encode natural images into a sequence of codes. The technique works by first passing the input vector (or image) through an encoder that reduces the dimensionality. The smaller dimensional vector is partitioned and each partition is quantized separately, thus resulting in a sequence of codes: one code per partition. These codes are then used by a decoder to recreate the original vector (or image).

RQ-VAE [20, 42] applies residual quantization to the output of the encoder of VQ-VAE to achieve a lower reconstruction error. We discuss this technique in more detail in Subsection 3.1.

Locality Sensitive Hashing (LSH) [13, 14] is a popular technique used for clustering and approximate nearest neighbor search. The particular version that we use in this paper for clustering is SimHash [1], which uses random hyperplanes to create binary vectors which serve as hashes of the items. Because it has low computational complexity and is scalable [13], we use this as a baseline technique for vector quantization.

3 PROPOSED FRAMEWORK

Our proposed technique consists of two components:

(1) *Semantic ID generation using content features.* This involves mapping the item content features to embedding vectors, which are further quantized into a tuple of semantic codewords. We call this tuple the item's Semantic ID.
(2) *Training a generative recommender system on Semantic IDs.* A Transformer based sequence-to-sequence model is trained on the sequences of semantic IDs corresponding to the items in the user's interaction history to predict the Semantic ID of the next item in the sequence.

Next, we explain these components in detail.

3.1 Semantic ID Generation

In this section, we describe the Semantic ID generation process for the items in the recommendation corpus. We assume that each item has associated content features that capture useful semantic information (*e.g.* titles or descriptions or images). Moreover, we assume that we have access to a pre-trained content encoder to generate a semantic embedding $\boldsymbol{x} \in \mathbb{R}^d$. For example, general-purpose pre-trained text encoders such as Sentence-T5 [25] and BERT [7] can be used to convert an item's text description to embeddings. In this work, we use the textual description of items as content features and use Sentence-T5 [25] encoder on this textual description. The semantic embeddings are then quantized to generate a Semantic ID for each item. Figure 2a gives a high-level overview of the process. This approach is similar to [12], where Hou *et al.* generate embeddings with a BERT encoder but the way they quantize the embeddings is different from ours.

We define a Semantic ID to be a tuple of codewords of length m. Each entry of the tuple, that is, each codeword can come from a different codebook. The number of items that the Semantic IDs can represent uniquely is thus equal to the product of the codebook sizes. While different techniques to generate Semantic IDs result in the codes having different semantic properties or guarantees, we want them to have the following property in general: *Similar items (items with similar content features or whose semantic embeddings are close) should have overlapping codewords.* For example, an item with Semantic ID $(10, 21, 35)$ should be more similar to one with Semantic ID $(10, 21, 40)$, than an item that is represented as $(10, 23, 32)$. Next, we discuss the quantization scheme RQ-VAE, which is used for Semantic ID generation.

3.1.1 RQ-VAE for Semantic IDs. Residual-Quantized Variational AutoEncoder (RQ-VAE) [42] is a multi-stage vector quantizer that applies quantization on residuals at multiple levels to generate a tuple of codewords (aka Semantic IDs). The Autoencoder jointly trains the codebook and the encoder-decoder to reconstruct the input using only Semantic IDs. Figure 3 illustrates the process of generating Semantic IDs through residual quantization.

RQ-VAE first encodes the input $\boldsymbol{x}$ via an encoder $\mathcal{E}$ to learn a latent representation $\boldsymbol{z} := \mathcal{E}(\boldsymbol{x})$. At the zeroth level ($d = 0$), the initial residual is simply defined as $\boldsymbol{r}_0 := \boldsymbol{z}$. At each level d, we have a codebook $\mathcal{C}_d := \{\boldsymbol{e}_k\}_{k=1}^K$, where K is the codebook size. Then, $\boldsymbol{r}_0$ is quantized by mapping it to the nearest embedding from that level's codebook. The index of the closest embedding $\boldsymbol{e}_{c_d}$ at $d = 0$, i.e., $c_0 = \arg\min_i \|\boldsymbol{r}_0 - \boldsymbol{e}_k\|$, represents the zeroth codeword. For the next level $d = 1$, the residual is defined as $\boldsymbol{r}_1 := \boldsymbol{r}_0 - \boldsymbol{e}_{c_0}$. Then, similar to the zeroth level, the code for the first level is computed by using the codebook for the first level. This process is repeated iteratively m times to get a tuple of m codewords that represent the Semantic ID. This recursive approach approximates the input from a coarse-to-fine granularity. Note that we chose to use a separate codebook of size K for each of the m levels, instead of using a single, m-times larger codebook. This design choice was motivated to avoid collisions between codewords at different granularity since the average norm of residuals decreases with increasing levels.

Once we have the Semantic ID $(c_0, \ldots, c_{m-1})$, the quantized representation of $\boldsymbol{z}$ is computed as $\widehat{\boldsymbol{z}} := \sum_{d=0}^{m-1} \boldsymbol{e}_{c_i}$. This vector, $\widehat{\boldsymbol{z}}$, is passed to the decoder, which tries to recreate the input $\boldsymbol{x}$ using $\widehat{\boldsymbol{z}}$.

The loss that we use to train the RQ-VAE is as follows:

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{x}) &:= \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{rqvae}}, \text{ where} \\
\mathcal{L}_{\text{recon}} &:= \|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|^2 \text{ and} \\
\mathcal{L}_{\text{rqvae}} &:= \sum_{d=0}^{m-1} \|\text{sg}[\boldsymbol{r}_i] - \boldsymbol{e}_{c_i}\|^2 + \beta\|\boldsymbol{r}_i - \text{sg}[\boldsymbol{e}_{c_i}]\|^2.
\end{aligned}
$$

Here $\widehat{\boldsymbol{x}}$ is the output of the decoder, and sg is the stop-gradient operation [34]. Note that this loss jointly trains the encoder, decoder, and the codebook.

As proposed in [42], to prevent RQ-VAE from a codebook collapse, where most of the input gets mapped to only a few codebook vectors, we use k-means clustering-based initialization for the codebook. More specifically, we apply the k-means algorithm on the first training batch and use the centroids as initialization.

3.1.2 Other options for quantization. A simple alternative to generating Semantic IDs is to use Locality Sensitive Hashing (LSH). We perform an ablation study in Section 4.3.2 where we find that RQ-VAE indeed works better than LSH. Another option is to use k-means clustering hierarchically [33], but it loses semantic meaning between different clusters [36]. We also tried VQ-VAE, and while it performs similarly to RQ-VAE for generating the candidates during retrieval, it loses the hierarchical nature of the IDs which confers many new capabilities as discussed in Section 4.4.

3.1.3 Handling Collisions. For retrieval, we would like to avoid collisions and assign unique IDs for items. Depending on the distribution of semantic embeddings, the choice of codebook size, and the length of codewords, collisions may still occur. For the hyperparameters we used (described in Section 4.1.3), we do observe a few items in the dataset with very similar semantic embeddings have the same Semantic ID assigned to them.

To remove the collisions we append an extra token at the end of the Semantic IDs to make them unique. For example, if two items share the Semantic ID $(12, 24, 52)$, we append extra tokens to differentiate between them, hence they are represented as $(12, 24, 52, 0)$ and $(12, 24, 52, 1)$.

3.2 Generative Retrieval with Semantic IDs

We construct item sequences for every user by sorting chronologically the items they have interacted with. Then, given a sequence of the form $(\text{item}_1, \ldots, \text{item}_n)$, the recommender system's task is to predict the next item item_{n+1}. For this, we propose a generative approach that directly predicts the Semantic IDs of items.

Formally, let $(c_{i,0}, \ldots, c_{i,m-1})$ be the m-length Semantic ID for item_i. Then, we convert the item sequence to the sequence $(c_{1,0}, \ldots, c_{1,m-1}, c_{2,0}, \ldots, c_{2,m-1}, \ldots, c_{n,0}, \ldots, c_{n,m-1})$. The sequence-to-sequence model is then trained to predict the Semantic ID of item_{n+1}, which is $(c_{n+1,0}, \ldots, c_{n+1,m-1})$. Hence, this formulation does not need to make any major modifications to existing sequence-to-sequence model architectures to train them for generative recommendations. Once we have the predicted tuple of codewords $(c_{n+1,0}, \ldots, c_{n+1,m-1})$, we simply look up the item to which this Semantic ID corresponds to. There is a possibility that the generated Semantic ID does not match any item in the dataset. However, as we observe in Figure 6, the probability of such an event is very low.

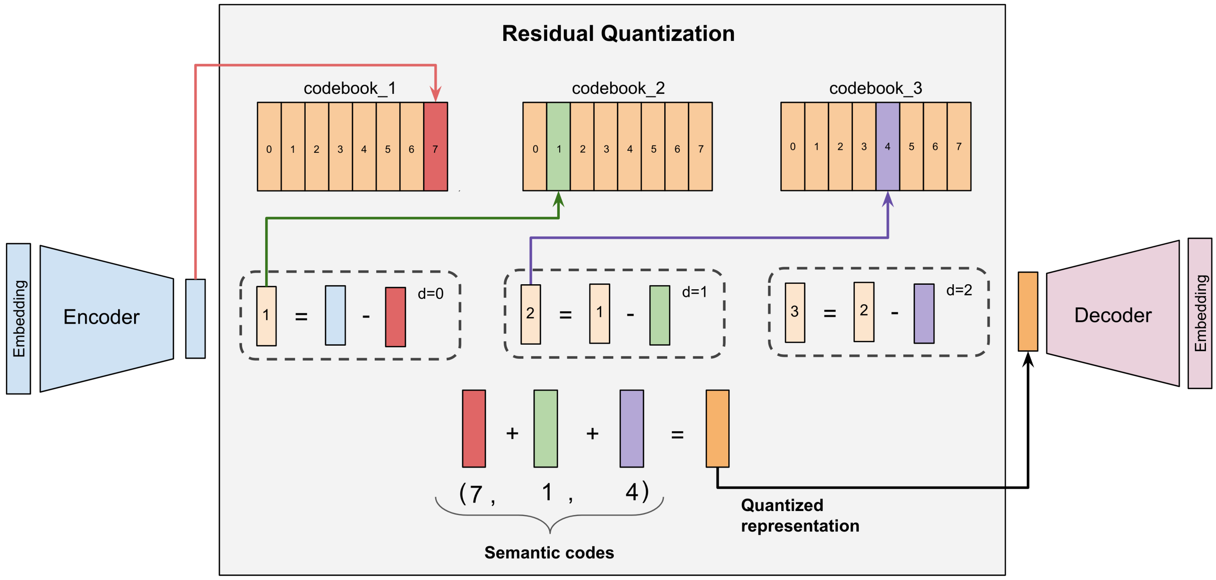


Figure 3: RQ-VAE: In the figure, the vector output by the DNN Encoder, say r_0 (represented by the blue bar), is fed to the quantizer, which works iteratively. First, the closest vector to r_0 is found in the first level codebook. Let this closest vector be e_{c_0} (represented by the red bar). Then, the residual error is computed as $r_1 := r_0 - e_{c_0}$. This is fed into the second level of the quantizer, and the process is repeated: The closest vector to r_1 is found in the second level, say e_{c_1} (represented by the green bar), and then the second level residual error is computed as $r_2 = r_1 - e'_{c_1}$. Then, the process is repeated for a third time on r_2. The semantic codes are computed as the indices of e_{c_0}, e_{c_1}, and e_{c_2} in their respective codebooks. In the example shown in the figure, this results in the code $(7, 1, 2)$.

4 EXPERIMENTS

We conduct exhaustive experiments to answer the following research questions (RQs):

- **RQ1**: How our proposed framework (TIGER) performs on the sequential recommendation task compared to the baseline methods?
- **RQ2**: Is the choice of our item representation with Semantic IDs meaningful?
- **RQ3**: What new recommendation capabilities emerge with this new paradigm?

4.1 Experimental Setup

In this section, we describe the datasets, evaluation metrics, and implementation details of the TIGER framework.

4.1.1 Datasets. We evaluate the proposed framework on three public real-world benchmarks from the Amazon Product Reviews dataset [10], containing user reviews and item metadata from May 1996 to July 2014. In particular, we use three categories of the Amazon Product Reviews dataset for the sequential recommendation task: "Beauty", "Sports and Outdoors", and "Toys and Games". Table 2 summarizes the statistics of the datasets. We use users' review history to create item sequences sorted by timestamp and filter out users with less than 5 reviews. Following the standard evaluation protocol [8, 17], we use the leave-one-out strategy for evaluation. For each item sequence, the last item is used for testing, the item before the last is used for validation, and the rest is used for training. During training, we limit the number of items in a user's history to 20.

4.1.2 Evaluation Metrics. We use top-k Recall (Recall@K) and Normalized Discounted Cumulative Gain (NDCG@K) with $K = 5, 10$ to evaluate the recommendation performance.

4.1.3 Implementation Details. Here we discuss the implementation details of the RQ-VAE model and the sequence-to-sequence model. We will release the source code of TIGER upon acceptance.

RQ-VAE Model. As discussed in section 3.1.1, RQ-VAE is used to quantize the semantic embedding of an item. We use the pre-trained Sentence-T5 model to obtain the semantic embedding of each item in the dataset. In particular, we use item's content features such as title, price, brand, and category to construct a sentence, which is then passed to the pre-trained Sentence-T5 model to obtain the item's semantic embedding of 768 dimension.

The RQ-VAE model consists of three components: a DNN encoder that encodes the input semantic embedding into a latent representation, residual quantizer which outputs a quantized representation, and a DNN decoder that decodes the quantized representation back to the semantic input embedding space. The encoder has three intermediate layers of size 512, 256 and 128 with ReLU activation, with a final latent representation dimension of 32. To quantize this representation, three levels of residual quantization is done. For each stage of the quantization, a codebook of cardinality 256 is maintained, where each vector in the codebook has a dimension of 32. When computing the total loss, we use $\beta = 0.25$. The RQ-VAE model is trained for 20k epochs to ensure high codebook usage ($\geq 80\%$). We use Adagrad optimizer with a learning rate of 0.4 and a batch size of 1024. Upon training, we use the learned encoder and the quantization module to generate a 3-tuple Semantic ID for each

Table 1: Performance comparison on the sequential recommendation task. The last row depicts the improvement observed with TIGER relative to the best baseline. We use bold and underline to denote the best and the second-best metric.

	Methods	Sports and Outdoors				Beauty				Toys and Games			
		Recall@5	NDCG@5	Recall@10	NDCG@10	Recall@5	NDCG@5	Recall@10	NDCG@10	Recall@5	NDCG@5	Recall@10	NDCG@10
Baselines	P5 [44]	0.0061	0.0041	0.0095	0.0052	0.0163	0.0107	0.0254	0.0136	0.0070	0.0050	0.0121	0.0066
	Caser [32]	0.0116	0.0072	0.0194	0.0097	0.0205	0.0131	0.0347	0.0176	0.0166	0.0107	0.0270	0.0141
	HGN [24]	0.0189	0.0120	0.0313	0.0159	0.0325	0.0206	0.0512	0.0266	0.0321	0.0221	0.0497	0.0277
	GRU4Rec [11]	0.0129	0.0086	0.0204	0.0110	0.0164	0.0099	0.0283	0.0137	0.0097	0.0059	0.0176	0.0084
	BERT4Rec [30]	0.0115	0.0075	0.0191	0.0099	0.0203	0.0124	0.0347	0.0170	0.0116	0.0071	0.0203	0.0099
	FDSA [44]	0.0182	0.0122	0.0288	0.0156	0.0267	0.0163	0.0407	0.0208	0.0228	0.0140	0.0381	0.0189
	SASRec [17]	0.0233	0.0154	0.0350	0.0192	0.0387	0.0249	0.0605	0.0318	0.0463	0.0306	0.0675	0.0374
	S^3-Rec [46]	0.0251	0.0161	0.0385	0.0204	0.0387	0.0244	0.0647	0.0327	0.0443	0.0294	0.0700	0.0376
	TIGER [Ours]	**0.0264**	**0.0181**	**0.0400**	**0.0225**	**0.0454**	**0.0321**	**0.0648**	**0.0384**	**0.0521**	**0.0371**	**0.0712**	**0.0432**
		+5.22%	+12.55%	+3.90%	+10.29%	+17.31%	+29.04%	+0.15%	+17.43%	+12.53%	+21.24%	+1.71%	+14.97%

Table 2: Dataset statistics for the three real-world benchmarks.

Dataset	# Users	# Items	Sequence Length	
			Mean	Median
Beauty	22,363	12,101	8.87	6
Sports and Outdoors	35,598	18,357	8.32	6
Toys and Games	19,412	11,924	8.63	6

item. To avoid multiple items being mapped to the same Semantic ID, we add a unique 4^{th} code for items that share the same first three codewords, *i.e.* two items associated with a tuple (7, 1, 4) are assigned (7, 1, 4, 0) and (7, 1, 4, 1) respectively (if there are no collisions, we still assign 0 as the fourth codeword). This results in a unique Semantic ID of length 4 for each item in the recommendation corpus. This is the Semantic ID generation algorithm that we use in TIGER. Notably, we observe less than 40 items sharing the same first three codewords, and hence the cardinality of the 4^{th} codeword is at most 40.

Sequence-to-Sequence Model. We use the open-sourced T5X framework [28] to implement our transformer based encoder-decoder architecture. To allow the model to process the input for the sequential recommendation task, we add semantic codewords to the vocabulary of the sequence-to-sequence model. In particular, we add 1024 (256×4) tokens to the vocabulary. In addition to the semantic codewords, we also add user-specific tokens to the vocabulary. To keep the vocabulary size limited, we only added 2000 tokens for user IDs. We use the Hashing Trick [37] to map the raw user ID to one of the 2000 user ID tokens. We construct the input sequence as the user Id token followed by the sequence of Semantic ID tokens corresponding to a given user's item interaction history. We found that adding user ID to the input, allows the model to personalize the recommended items.

We use 4 layers each for the encoder and decoder models with 6 self-attention heads of dimension 64 in each layer. We used the ReLU activation function for all the layers. The MLP and the input dimension was set as 1024 and 128, respectively. We used a dropout of 0.1. Overall, the model has around 13 million parameters. We train this model for 200k steps for the "Beauty" and "Sports and Outdoors" dataset. Due to the smaller size of the "Toys and Games" dataset, it is trained only for 100k steps. We use a batch size of 256. The learning rate is 0.01 for the first 10k steps and then follows an inverse square root decay schedule.

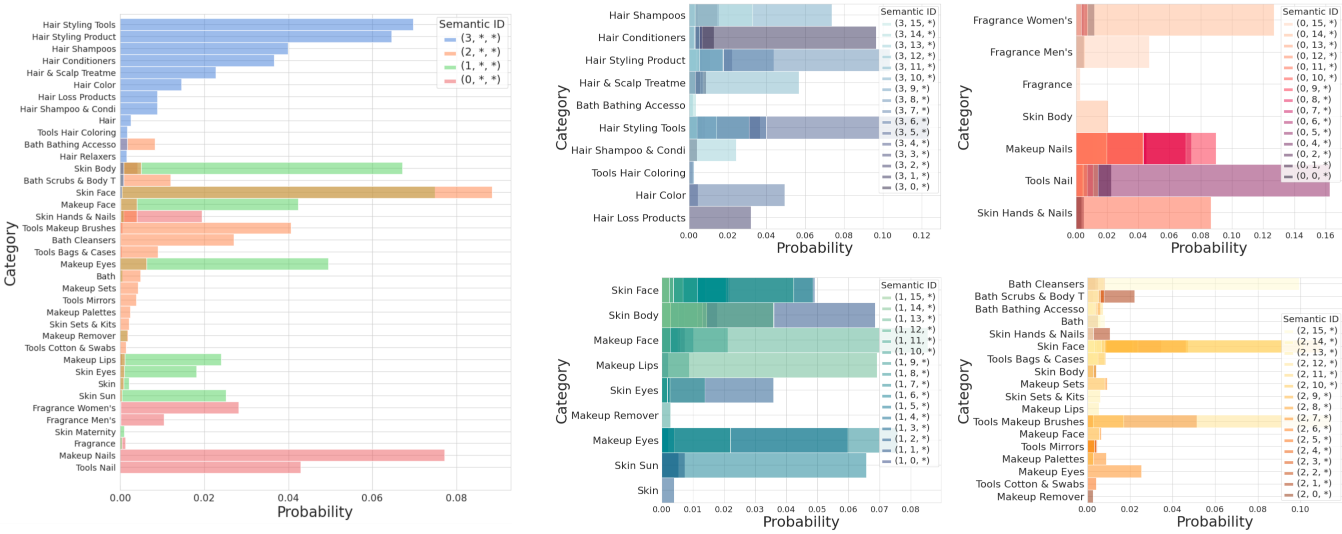
4.2 Performance on Sequential Recommendation (RQ1)

We first evaluate our model on the sequential recommendation task and compare the performance of TIGER against existing state-of-the-art sequential recommendation models. Next, we briefly describe the recent baselines proposed for the sequential recommendation task and discuss the performance of the all the models.

4.2.1 Baselines. We compare our proposed framework for generative retrieval with several other sequential recommendation methods.

- GRU4Rec [11] is the first RNN-based approach that uses a customized GRU for the sequential recommendation task.
- Caser [32] uses a CNN architecture for capturing high-order Markov Chains by applying horizontal and vertical convolutional operations for sequential recommendation.
- HGN [24]: Hierarchical Gating Network (HGN) captures the long-term and short-term user interests via a new gating architecture.
- SASRec [17]: Self-Attentive Sequential Recommendation (SASRec) uses a causal mask Transformer to model a user's sequential interactions.
- BERT4Rec [30]: BERT4Rec addresses the limitations of unidirectional architectures by using a bi-directional self-attention Transformer for the recommendation task.
- FDSA [44]: Feature-level Deeper Self-Attention Network (FDSA) incorporates item features in addition to the item embeddings as part of the input sequence in the Transformers.
- S^3-Rec [46]: Self-Supervised Learning for Sequential Recommendation (S^3-Rec) proposes pre-training a bi-directional Transformer on self-supervision tasks to improve the sequential recommendation.
- P5 [8]: P5 is a recent method that uses a pretrained Large Language Model (LLM) to unify different recommendation tasks in a single model.

Notably all the baselines, with the exception of P5, learn a high-dimensional vector space using dual encoder, where the user's past item interactions and the candidate items are encoded as a high-dimensional representation and Maximum Inner Product Search (MIPS) is used to retrieve the next candidate item that the user will potentially interact with. In contrast, our *novel* generative retrieval framework directly predicts the item's Semantic ID token-by-token using a sequence-to-sequence model.



(a) **The ground-truth category distribution for all the items in the dataset colored by the value of the first codeword c_1.**

(b) **The ground-truth category distributions for all the items having the Semantic ID as $(c_1, *, *)$, where $c_1 \in \{1, 2, 3, 4\}$. The categories are color-coded based on the second semantic token c_2.**

Figure 4: Qualitative study of RQ-VAE Semantic IDs (c_1, c_2, c_3, c_4) on the Amazon Beauty dataset. We show that the ground-truth categories are distributed across different Semantic tokens. Moreover, the RQVAE semantic IDs form a hierarchy of items, where the first semantic token (c_1) corresponds to coarse-level category, while second/third semantic token (c_2/c_3) correspond to fine-grained categories.

4.2.2 *Recommendation Performance.*

We perform an extensive analysis of our proposed TIGER on the sequential recommendation task and compare against several recent baselines. The results for all baselines, except P5, are taken from the publicly accessible results [1] made available by Zhou *et al.* [46]. For P5, we use the source code[2] made available by the authors. However, for a fair comparison, we updated the data pre-processing method to be consistent with the other baselines and our method. We provide further details related to our changes in Appendix A.

The results are shown in Table 1. We observe that TIGER consistently outperforms the existing baselines. We see significant improvement across all the three benchmarks that we considered. In particular, TIGER performs considerably better on the Beauty benchmark compared to the second-best baseline with up to 29% improvement in NDCG@5 compared to SASRec and 17.3% improvement in Recall@5 compared to S^3-Rec. Similarly on the Toys and Games dataset, TIGER is 21% and 15% better in NDCG@5 and NDCG@10, respectively.

4.3 Item Representation (RQ2)

In this section, we analyze several important characteristics of RQ-VAE Semantic IDs. In particular, we first perform a qualitative analysis to observe the hierarchical nature of Semantic IDs in Section 4.3.1. Next, we evaluate the importance of our design choice of using RQ-VAE for quantization by contrasting the performance with an alternative hashing-based quantization method in section 4.3.2. Finally, we perform an ablation in section 4.3.3 to study the importance of using Semantic IDs by comparing TIGER with a sequence-to-sequence model that uses Random ID for item representation.

4.3.1 *Qualitative Analysis.*

We analyze the RQ-VAE Semantic IDs learned for the Amazon Beauty dataset in Figure 4. For exposition, we set the number of RQ-VAE levels as 3 with a codebook size of 4, 16, and 256 respectively, *i.e.* for a given Semantic ID (c_1, c_2, c_3) of an item, $0 \le c_1 \le 3$, $0 \le c_2 \le 15$ and $0 \le c_3 \le 255$.

In Figure 4a, we annotate each item's category using c_1 to visualize c_1-specific categories in the overall category distribution of the dataset. As shown in Figure 4a, c_1 captures the high-level category of the item. For instance, $c_1 = 3$ contains most of the products related to "Hair". Similarly, majority of items with $c_1 = 1$ are "Makeup" and "Skin" products for face, lips and eyes.

We also visualize the hierarchical nature of RQ-VAE Semantic IDs by fixing c_1 and visualizing the category distribution for all possible values of c_2 in Figure 4b. Once again, we found that the second codeword c_2 can further categorize the high-level semantics captured with c_1 into fine-grained categories.

The hierarchical nature of Semantic IDs learned by RQ-VAE opens a wide-array of new capabilities which are discussed in Section 4.4. As opposed to existing recommendation systems that learn item embeddings based on random atomic IDs, TIGER uses Semantic IDs where semantically similar items have overlapping codewords, which allows the model to effectively share knowledge from semantically similar items in the dataset.

4.3.2 *Hashing vs. RQ-VAE for Semantic ID Generation.*

In this section, we study the importance of RQ-VAE in our framework by comparing RQ-VAE against Locality Sensitive Hashing (LSH) [1, 13, 14] for Semantic ID generation. LSH is a popular hashing technique that can be easily adapted to work for our setting. To generate LSH Semantic IDs, we use h random hyperplanes $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_h$ to perform a random projection of the embedding vector $\boldsymbol{x}$ and compute the

[1] https://github.com/aHuiWang/CIKM2020-S3Rec
[2] https://github.com/jeykigung/P5

Table 3: Ablation study for different ID generation techniques for generative retrieval. We show that RQ-VAE Semantic IDs perform significantly better compared to hashing-based Semantic IDs and Random IDs.

Methods	Sports and Outdoors				Beauty				Toys and Games			
	Recall@5	NDCG@5	Recall@10	NDCG@10	Recall@5	NDCG@5	Recall@10	NDCG@10	Recall@5	NDCG@5	Recall@10	NDCG@10
Random ID	0.007	0.005	0.0116	0.0063	0.0296	0.0205	0.0434	0.0250	0.0362	0.0270	0.0448	0.0298
LSH Semantic ID	0.0215	0.0146	0.0321	0.0180	0.0379	0.0259	0.0533	0.0309	0.0412	0.0299	0.0566	0.0349
RQ-VAE Semantic ID	**0.0264**	**0.0181**	**0.0400**	**0.0225**	**0.0454**	**0.0321**	**0.0648**	**0.0384**	**0.0521**	**0.0371**	**0.0712**	**0.0432**

following binary vector: $(1_{w_1^\top x>0}, \ldots, 1_{w_h^\top x>0})$. This vector is converted into an integer code as $c_0 = \sum_{i=1}^{h} 2^{i-1} 1_{w_i^\top x>0}$. This process is repeated m times using an independent set of random hyperplanes, resulting in m codewords $(c_0, c_1, \ldots, c_{m-1})$, which we refer to as the LSH-based Semantic ID.

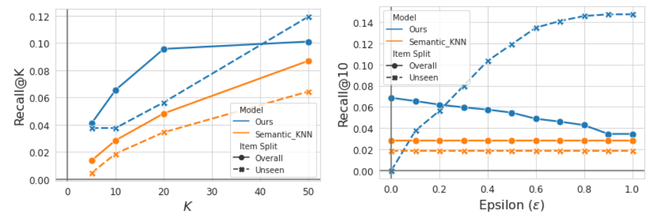
In Table 3, we compare the performance of LSH Semantic ID with our proposed RQ-VAE Semantic ID. In this experiment, for LSH Semantic IDs, we used $h = 8$ random hyperplanes and set $m = 4$ to be comparable against RQ-VAE in terms of cardinalities. The parameters for the hyperplanes are randomly sampled from a standard normal distribution, which ensures that the hyperplanes are spherically symmetric. Our results show that RQ-VAE consistently outperforms LSH. This illustrates that learning Semantic IDs via a non-linear, Deep Neural Network (DNN) architecture yields better quantization than using random projections, given the same content-based semantic embedding.

4.3.3 Random ID vs. Semantic ID. We also compare the importance of Semantic IDs in our generative retrieval recommender system. In particular, we compare randomly generated IDs with the Semantic IDs. To generate the Random ID baseline, we assign m random codewords to each item. A Random ID of length m for an item is simply $(c_1, \ldots, c_m)$, where c_i is sampled uniformly at random from $\{1, 2, \ldots, K\}$. We set $m = 4$, and $K = 255$ for the Random ID baseline to make the cardinality similar to RQ-VAE Semantic IDs. A comparison of Random ID against RQ-VAE and LSH Semantic IDs is shown in Table 3. We see that Semantic IDs consistently outperform Random ID baseline, highlighting the importance of leveraging content-based semantic information.

4.4 New Capabilities (RQ3)

We describe two new capabilities that directly follow from our proposed generative retrieval framework, namely cold-start recommendations and recommendation diversity. We refer to these capabilities as "new" since existing sequential recommendation models (See Section 4.2.1) cannot be directly used to satisfy these real-world use cases. We believe these capabilities result from a synergy between RQ-VAE based Semantic IDs and the generative retrieval approach of our framework. We discuss how TIGER is used in these settings in the following sections.

4.4.1 Cold-Start Recommendation. In this section, we study the cold-start recommendation capability of our proposed framework. Due to the fast-changing nature of the real-world recommendation corpus, new items are constantly introduced. Since newly added items lack user impressions in the training corpus, existing recommendation models that use a random atomic ID for item representation fail to retrieve new items as potential candidates. In contrast, TIGER can easily perform cold-start recommendations in an end-to-end fashion.



(a) Recall@K vs. K, when $\epsilon = 0.1$. **(b) Recall@10 vs. ϵ.**

Figure 5: Performance in the cold-start retrieval setting.

For this analysis, we consider the Beauty dataset from Amazon Reviews. To simulate newly added items, we remove 5% of test items from the training data split. We refer to these removed items as *unseen items*. Removing the items from the training split ensures there is no data leakage with respect to the unseen items.

As before, we use Semantic ID of length 4 to represent the items, where the first 3 tokens are generated using RQ-VAE and the 4^{th} token is used to ensure a unique ID exists for all the seen items. We train the RQ-VAE quantizer and the sequence-to-sequence model on the training split. Once trained, we use the RQ-VAE model to generate the Semantic IDs for all the items in the dataset, including any unseen items in the item corpus.

Given a Semantic ID (c_1, c_2, c_3, c_4) predicted by the model, we retrieve the seen item having the same corresponding ID. Note that by definition, each Semantic ID predicted by the model can match at most one item in the training dataset. Additionally, unseen items having the same first three semantic tokens, *i.e.* (c_1, c_2, c_3) are included to the list of retrieved candidates. Finally, when retrieving a set of top-K candidates, we introduce a hyperparameter ϵ which specifies the maximum proportion of unseen items chosen by our framework.

We compare the performance of TIGER with a k-Nearest Neighbors (KNN) approach on the cold-start recommendation setting in Figure 5. For KNN, we use the semantic representation space to perform the nearest-neighbor search. We refer to the KNN-based baseline as Semantic_KNN. Figure 5a shows that our framework with $\epsilon = 0.1$ consistently outperforms Semantic_KNN for all Recall@K metrics. In Figure 5b, we provide a comparison between our method and Semantic_KNN for various values of ϵ. For all settings of $\epsilon \geq 0.1$, our method outperforms the baseline.

4.4.2 Recommendation diversity. While Recall and NDCG are the primary metrics used to evaluate a recommendation system, diversity of predictions is another critical objective of interest. A recommender system with poor diversity can be detrimental to the long-term engagement of users. In this section, we discuss how our generative retrieval framework can be used to predict diverse items. We show that temperature-based sampling during the decoding process can be effectively used to control the diversity of model

Table 4: Recommendation diversity with temperature-based decoding using generative retrieval.

Target Category	Most-common Categories for top-10 predicted items	
	T = 1.0	T = 2.0
Hair Styling Products	Hair Styling Products	Hair Styling Products, Hair Styling Tools, Skin Face
Tools Nail	Tools Nail	Tools Nail, Makeup Nails
Makeup Nails	Makeup Nails	Makeup Nails, Skin Hands & Nails, Tools Nail
Skin Eyes	Skin Eyes	Hair Relaxers, Skin Face, Hair Styling Products, Skin Eyes
Makeup Face	Tools Makeup Brushes, Makeup Face	Tools Makeup Brushes, Makeup Face, Skin Face, Makeup Sets, Hair Styling Tools
Hair Loss Products	Hair Loss Products, Skin Face, Skin Body	Skin Face, Hair Loss Products, Hair Shampoos, Hair & Scalp Treatments, Hair Conditioners

Table 5: Recommendation diversity in the model predictions. We evaluate the entropy of the distribution of categories that the model predicts in the Amazon Beauty dataset. A higher entropy corresponds more diverse items predicted by the model.

Temperature	Entropy@10	Entropy@20	Entropy@50
T = 1.0	0.76	1.14	1.70
T = 1.5	1.14	1.52	2.06
T = 2.0	1.38	1.76	2.28

predictions. While temperature-based sampling can be applied to any existing recommendation model, TIGER allows sampling across various levels of hierarchy owing to the properties of RQ-VAE Semantic IDs. For instance, sampling the first token of the Semantic ID allows retrieving items from coarse-level categories, while sampling a token from second/third token allows sampling items within a category.

We quantitatively measure the diversity of predictions using Entropy@K metric, where the entropy is calculated for the distribution of the ground-truth categories of the top-K items predicted by the model. We report the Entropy@K for various temperature values in Table 5. We observe that temperature-sampling in the decoding stage can be effectively used to increase the diversity in the ground-truth categories of the items. We also perform a qualitative analysis in Table 4.

5 DISCUSSION

Invalid IDs. Since the model decodes the codewords of the target Semantic ID autoregressively, it is possible that the model can predict an invalid ID (i.e., it may not map to any item in the recommendation dataset). In our experiments, we used semantic IDs of length 4 with each codeword having a cardinality of 256 (i.e., codebook size = 256 for each level). The number of possible IDs spanned by this combination = 256^4, which is approx. 4 trillion. On the other hand, the number of items in the datasets we consider is 10K-20K (See Table 2). Even though the number of valid IDs is only a fraction of all complete ID space, we observe that the model almost always predicts the valid IDs. We visualize the fraction of invalid IDs produced by TIGER as a function of the number of retrieved items K in Figure 6.

Effects of Semantic ID length and codebook size. We tried varying the Semantic ID length and codebook size, such as having an ID consisting of 6 codewords each from a codebook of size 64. We noticed that the recommendation metrics for TIGER were robust to these changes. However, note that the input sequence length increases with longer IDs (i.e., more codewords per ID), which makes the computation more expensive for our transformer-based

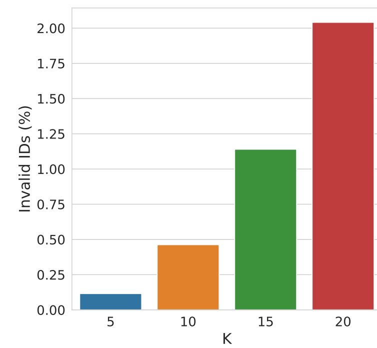


Figure 6: % of invalid IDs when generating Semantic IDs using Beam search for various values of K for the Beauty dataset. As shown, only $\tilde{2}$% of the IDs are invalid when retrieving the top-20 items.

sequence-to-sequence model.

Inference cost. Despite the remarkable success of our model on the sequential recommendation task, we note that our model is more expensive than ANN-based models during inference (not during training) due to the use of beam search for autoregressive decoding. We emphasize that optimizing the computational efficiency of TIGER was not the main objective of this work. Instead, our work opens up a new area of research: *Recommender Systems based on Generative Retrieval*. As part of future work, we will consider ways to make the model smaller or explore other ways of leveraging Transformer's capabilities, such as building a unified model across multiple datasets and tasks.

6 CONCLUSION

This paper proposed a novel paradigm, called TIGER, to retrieve candidates in recommender systems using a generative model. Underpinning this method is a novel semantic ID representation for items, which uses a hierarchical quantizer (RQ-VAE) on content embeddings to generate tokens that form the semantic IDs. Our method is end-to-end, i.e., a single model can be used to train and serve without creating an index — the transformer memory acts as the index [33]. We note that the cardinality of our embedding table does not grow linearly with the cardinality of item space, which works in our favor compared to systems that need to create large embedding tables during training or generate an index for every single item. Through experiments on three datasets, we show that our model can achieve SOTA results and generalizes well to new and unseen items.

Our work enables many new research directions. For example, it will be interesting to explore how these Semantic IDs can be integrated with LLMs to enable much more powerful conversational recommender models. We will also explore how to improve the Semantic ID representation and how to use it for ranking.

REFERENCES

[1] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[2] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[3] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[4] Zeyu Cui, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. M6-rec: Generative pretrained language models are open-ended recommender systems. *arXiv preprint arXiv:2205.08084*, 2022.

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

[6] 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 *Fifteenth ACM Conference on Recommender Systems*, pages 143–153, 2021.

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

[8] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). *arXiv preprint arXiv:2203.13366*, 2022.

[9] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19, 2015.

[10] Ruining He and Julian McAuley. 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*, pages 507–517, 2016.

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

[12] Yupeng Hou, Zhankui He, Julian McAuley, and Wayne Xin Zhao. Learning vector-quantized item representation for transferable sequential recommenders. *arXiv preprint arXiv:2210.12316*, 2022.

[13] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[14] Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 618–625, 1997.

[15] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[16] Wang-Cheng Kang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H Chi. Learning to embed categorical features without embedding tables for recommendation. *arXiv preprint arXiv:2010.10784*, 2020.

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

[18] Dongmoon Kim, Kun-su Kim, Kyo-Hyun Park, Jee-Hyong Lee, and Keon Myung Lee. A music recommendation system with a dynamic k-means clustering algorithm. In *Sixth international conference on machine learning and applications (ICMLA 2007)*, pages 399–403. IEEE, 2007.

[19] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[20] 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*, pages 11523–11532, 2022.

[21] Hyunji Lee, Jaeyoung Kim, Hoyeon Chang, Hanseok Oh, Sohee Yang, Vlad Karpukhin, Yi Lu, and Minjoon Seo. Contextualized generative retrieval. *arXiv preprint arXiv:2210.02068*, 2022.

[22] Hyunji Lee, Sohee Yang, Hanseok Oh, and Minjoon Seo. Generative retrieval for long sequences. *arXiv preprint arXiv:2204.13596*, 2022.

[23] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1419–1428, 2017.

[24] Chen Ma, Peng Kang, and Xue Liu. Hierarchical gating networks for sequential recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 825–833, 2019.

[25] Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith Hall, Daniel Cer, and Yinfei Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1864–1874, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[26] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[27] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

[28] Adam Roberts, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, Curtis Hawthorne, Aitor Lewkowycz, Alex Salcianu, Marc van Zee, Jacob Austin, Sebastian Goodman, Livio Baldini Soares, Haitang Hu, Sasha Tsvyashchenko, Aakanksha Chowdhery, Jasmijn Bastings, Jannis Bulian, Xavier Garcia, Jianmo Ni, Andrew Chen, Kathleen Kenealy, Jonathan H. Clark, Stephan Lee, Dan Garrette, James Lee-Thorp, Colin Raffel, Noam Shazeer, Marvin Ritter, Maarten Bosma, Alexandre Passos, Jeremy Maitin-Shepard, Noah Fiedel, Mark Omernick, Brennan Saeta, Ryan Sepassi, Alexander Spiridonov, Joshua Newlan, and Andrea Gesmundo. Scaling up models and data with `t5x` and `seqio`. *arXiv preprint arXiv:2203.17189*, 2022.

[29] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[30] 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*, pages 1441–1450, 2019.

[31] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[32] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 565–573, 2018.

[33] Yi Tay, Vinh Q Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, et al. Transformer memory as a differentiable search index. *arXiv preprint arXiv:2202.06991*, 2022.

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

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[36] Yujing Wang, Yingyan Hou, Haonan Wang, Ziming Miao, Shibin Wu, Hao Sun, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, et al. A neural corpus indexer for document retrieval. *arXiv preprint arXiv:2206.02743*, 2022.

[37] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1113–1120, 2009.

[38] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[39] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 269–277, 2019.

[40] Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved VQGAN. In *International Conference on Learning Representations*, 2022.

[41] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, et al. Scaling autoregressive models for content-rich text-to-image generation. *arXiv preprint arXiv:2206.10789*, 2022.

[42] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. Soundstream: An end-to-end neural audio codec. *CoRR*, abs/2107.03312, 2021.

[43] Shuai Zhang, Yi Tay, Lina Yao, and Aixin Sun. Next item recommendation with self-attention. *arXiv preprint arXiv:1808.06414*, 2018.

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

[45] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 43–51, 2019.

[46] 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*, pages 1893–1902, 2020.

A MODIFICATIONS TO THE P5 DATA PREPROCESSING

The P5 source code [3] pre-processes the Amazon dataset to first create sessions for each user containing the chronologically ordered list of items the user reviewed. After creating these sessions, the original item IDs from the dataset are remapped to integers $1, 2, 3, \ldots$ [4]. Hence, the first item in the first session gets an id of '1', the second item, if not seen before, gets an id of '2', and so on. This results in the creation of a sequential dataset where a lot of the sequences are of the form $a, a+1, a+2, \ldots$. Since LLMs like T5 [27] are known to be able to recognize simple consecutive integer sequences as above, this dataset introduces a bias which might result in higher metrics. Note that this dataset would not have been a problem for non-LLM based recommender systems since they do not intrinsically understand integer sequences. To remove this bias, instead of assigning sequentially increasing integer ids to items, we assigned them in a random manner, and then created sequence datasets for training and evaluation. The rest of the code for P5 was kept identical to the source code provided in the paper. The results for this dataset are reported in Table 6 in the row 'P5'. We also implemented a version of P5 ourselves from scratch, for only sequential recommendation task, whose results for the dataset described above are mentioned in the row 'P5-ours'.

We were also able to verify in our P5 implementation that using consecutive integer sequences for the item IDs helped us get equivalent or better metrics than those reported in P5.

Table 6: Results of P5[8] with changes to prevent consecutive integer sequences of the form described in Appendix A

Methods	Sports and Outdoors				Beauty				Toys and Games			
	Recall@5	NDCG@5	Recall@10	NDCG@10	Recall@5	NDCG@5	Recall@10	NDCG@10	Recall@5	NDCG@5	Recall@10	NDCG@10
P5	0.0061	0.0041	0.0095	0.0052	0.0163	0.0107	0.0254	0.0136	0.0070	0.0050	0.0121	0.0066
P5-ours	0.0107	0.0076	0.01458	0.0088	0.035	0.025	0.048	0.0298	0.018	0.013	0.0235	0.015

[3] https://github.com/jeykigung/P5
[4] https://github.com/jeykigung/P5/blob/0aaa3fd8366bb6e708c8b70708291f2b0ae90c82/preprocess/data_preprocess_amazon.ipynb