# TIGER: Generative Retrieval for Sequential Recommendation on Amazon Toys & Games

Kevin Bretz
Leiden University

Michail Protonotarios
Leiden University

Maftukhatul Q. Virati
Leiden University

## Abstract

In this study we focus on our implementation of Transformer Index for Generative Recommenders (TIGER) [Rajput et al.(2023)] for sequential recommendation on the Amazon Toys & Games 2014 dataset [McAuley et al.(2015)]. Each item is encoded as a tuple of discrete Semantic IDs by a Residual-Quantized VAE (RQ-VAE) and an encoder-decoder Transformer is trained to generate the Semantic ID of the next item one token at a time. Training was done on a single Google Colab T4 GPU. The default configuration reaches Recall@5 = 0.0189 and NDCG@5 = 0.0122 on the held-out test items and a dropout-tuned variant ($p$=0.2) reaches Recall@5 = 0.0227 (+20%). We also report ablation across the hyperparameters, a qualitative look at Semantic-ID prefix neighbourhoods and a cold-start experiment.

## 1 Introduction

Sequential recommendation usually relies on dense item embeddings and nearest-neighbour retrieval, the example of such case is SASRec [Kang and McAuley(2018)]. However, TIGER [Rajput et al.(2023)] takes a different approach where items are first turned into short tuples of discrete tokens (Semantic IDs) by an RQ-VAE on content embeddings and a sequence-to-sequence Transformer is trained to autoregressively generate the next item's Semantic ID. This made retrieval into a generation task rather than a vector-search task.

In this study, we focus on implemantion of both stages from scratch in PyTorch and evaluated on the 2014 Amazon Reviews Toys & Games subset [McAuley et al.(2015)] using the full-ranking leave-one-out protocol. We ran a ablation experiment across multiple hyperparameters. Additionally we also did dropout sweep and cold-start experiment.

## 2 Methodology

### 2.1 Dataset & preprocessing

We use both 2014 Toys & Games reviews and the metadata files. After running iterative 5-core filtering where we drop users and items with fewer than 5 interactions and repeated it until both counts stop changing. The total dataset are 19,412 users, 11,924 items and 167,597 interactions. The mean / median / max training-history length is 5.9 / 4 / 18 items.

Following TIGER, we then treat every review as an implicit positive and cap each user's history at 20 most-recent items. For splitting the dataset into training, validation and test, we use the leave-one-out split to put each user's last interaction in the test set, the second-to-last in the validation set and the rest in training set.

For each item, we concatenate *title, categories, brand, price, description* into one sentence and encode it with `sentence-transformers/sentence-t5-base` [Ni et al.(2022)] to a 768-dimensional content embedding. The embeddings then L2-normalised and cached on disk, we do this to reduce the computational load as we only compute them once.

### 2.2 RQ-VAE

The RQ-VAE has an MLP encoder 768→512→256→128→32 (ReLU), a residual quantiser with $L$=3 levels of $K$=256 codes each and a mirrored MLP decoder. At each level $l$ the residual is mapped to its nearest code $c_l$, the reconstruction is decoded from $\mathbf{z}_q = \sum_l \text{codebook}_l[c_l]$. Gradients are passed through the quantiser with the standard straight-through estimator $\mathbf{z}_q^{ST} = \mathbf{z} + (\mathbf{z}_q - \mathbf{z}).\text{detach}()$. The training loss is

$$\mathcal{L} = \text{MSE}(\hat{\mathbf{x}}, \mathbf{x}) + \beta \sum_l \text{MSE}(\mathbf{r}_l, \text{sg}[c_l])$$

with $\beta = 0.25$

We update the codebook entries with an exponential moving average over the residuals assigned to them (decay 0.99), similar with previous study in [van den Oord et al.(2017)], rather than via gradient descent on a codebook MSE loss. We also reset any code whose EMA cluster size drops below 2 by sampling a fresh residual from the current batch thus prevents collapse during early training. The Codebooks are initialised with 10 iterations of K-means on the residuals at each level.

After training we extract one $L$-tuple per item. For items that collide on the same tuple get an extra integer disambiguation suffix, so every item ends up with a unique $(L{+}1)$-tuple, following TIGER [Rajput et al.(2023)] implementation.

### 2.3 Generative Transformer

Each user's history is flattened into a sequence of Semantic-ID tokens, every item contributes $L{+}1 = 4$ tokens. Token IDs are level-prefixed where codebook level $l$ occupies $(lK, (l{+}1)K)$, suffixes occupy $(LK, LK{+}M{+}1)$ and `[PAD]`, `[BOS]`, `[EOS]` sit at the top. Then with $L$=3, $K$=256 and max suffix $M$=7 the vocabulary has 779 tokens.

The model in generative transformer is a standard encoder-decoder Transformer with 4 encoder layers, 4 decoder layers, 6 attention heads of dimension 64 ($d_{\text{model}}$=384), feed-forward inner dimension 1024 and dropout 0.1. The input embedding and the output projection also share weights. And last, the encoder is bidirectional, which means that the decoder is causal and cross-attends to the encoder.

Training uses AdamW with lr $3{\times}10^{-4}$, weight decay 0.01, betas $(0.9, 0.98)$. We use linear warmup over 4,000 steps then cosine decay to $10^{-5}$, fp16 mixed precision with `GradScaler`, gradient clipping at 0.5, batch size 256 and cross-entropy on the $L{+}2$ decoder positions

ignoring [PAD]. We also use early-stop on NDCG@10 validation with patience of 10.

## 2.4 Inference & evaluation

At inference time we run beam search with beam size 10 for $L$+1 decoding steps, ranking beams by cumulative log-probability. The resulting Semantic-ID tuples are looked up in a precomputed valid-tuple table which means that any beam whose tuple does not match a real item is counted as a miss and tracked as an *invalid generation*. We then report metric such as Recall@5, Recall@10, NDCG@5 and NDCG@10 on the test target, using the full-ranking protocol.

# 3 Experiments

## 3.1 RQ-VAE training

The default RQ-VAE was trained for 200 epochs. Early stopping is gated on both a recon-loss plateau and codebook-usage stability, since the recon loss flattens out well before usage stabilises.

We applied the codebooks to all 11,924 items at the end of training, with every code at every level is the nearest code for at least one item, so the assignment-level utilisation is 100% at all three levels. The collision rate is 4.30%: 513 items share their $(c_0, c_1, c_2)$ prefix with at least one other item and the largest suffix value needed is 7. Figure 1 shows the training curves. The right bar chart shows the lower in-batch usage at the end of training. The EMA only sees one minibatch at a time, so a single batch hits about 80% of the codebook, so the assignment-level 100% above is what matters for downstream token coverage.

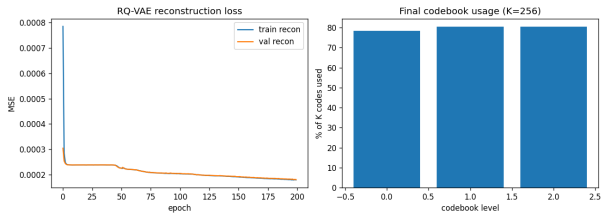


**Figure 1: RQ-VAE training. Left: per-epoch reconstruction MSE on train (blue) and val (orange), the two curves stay on top of each other, so there is no overfitting. Right: in-batch codebook usage at the end of training.**

## 3.2 Default-config Transformer

With the recommended default settings ($L$=3, $K$=256, $d_{\text{model}}$=384, 4+4 layers, 6 heads, dropout 0.1, beam 10). Using Google Colab T4, we training run for 77 minutes. Early stopping is triggered at epoch 33, with the top validation score happen at epoch 25. The final test metrics are R@5 = 0.0189, R@10 = 0.0293, N@5 = 0.0122, N@10 = 0.0156, with an invalid-generation rate of 0.89% at beam 10. These numbers are about 3× below the TIGER paper's reported scores on this benchmark. As we have much smaller training budget and the base-size sentence encoder, this is still reasonable. We also shows the training curves in Figure 2.

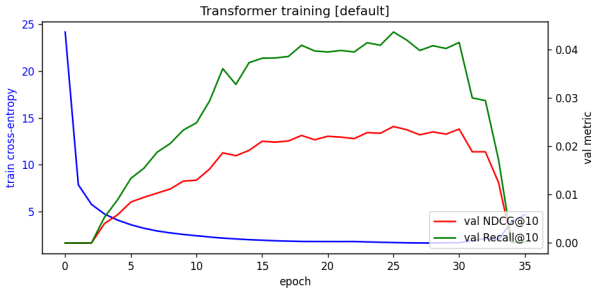


**Figure 2: Default-config Transformer training on Colab T4. Train cross-entropy (blue, left axis) decreases monotonically; val NDCG@10 (red) and Recall@10 (green) climb to a peak around epoch 25 and then drift down, after which early stopping ends the run at epoch 33.**

| Axis | Value | R@5 | R@10 | N@5 | N@10 | inv. % |
|---|---|---|---|---|---|---|
| **default** | $K$=256, $L$=3, $d$=384 | 0.0189 | 0.0293 | 0.0122 | 0.0156 | 0.9 |
| $K$ | 64 | 0.0129 | 0.0192 | 0.0083 | 0.0104 | 1.4 |
| $K$ | 128 | 0.0153 | 0.0249 | 0.0104 | 0.0135 | 0.9 |
| $L$ | 2 | 0.0158 | 0.0235 | 0.0104 | 0.0129 | 2.4 |
| $L$ | 4 | **0.0197** | **0.0325** | **0.0129** | **0.0170** | 0.9 |
| $d_{\text{model}}$ | 192 | **0.0199** | **0.0317** | **0.0124** | **0.0162** | 0.1 |
| $d_{\text{model}}$ | 768 | 0.0154 | 0.0219 | 0.0100 | 0.0121 | 11.0 |
| layers | 2 | **0.0191** | 0.0293 | **0.0127** | **0.0160** | 0.7 |
| layers | 6 | 0.0144 | 0.0226 | 0.0095 | 0.0121 | 1.4 |
| heads | 4 | 0.0179 | 0.0283 | 0.0116 | 0.0150 | 0.8 |
| heads | 8 | 0.0188 | **0.0298** | 0.0119 | 0.0154 | 0.7 |
| **dropout** | **0.2** | **0.0227** | **0.0352** | **0.0140** | **0.0181** | 0.5 |
| beam size | 5 | 0.0176 | 0.0176 | 0.0117 | 0.0117 | 0.6 |
| beam size | 20 | 0.0189 | **0.0304** | 0.0122 | **0.0159** | 1.6 |

**Table 1: One-axis-at-a-time ablation results. Bold entries beat the default on that metric. The strongest single configuration is dropout 0.2 (+20% on R@5).**

## 3.3 Ablations

For the ablation we ran a one-axis-at-a-time ablation across every axis for $K$, $L$, $d_{\text{model}}$, number of layers, number of heads, beam size and dropout sweep, which turn to had a large effect. The full numerical results are shown in the Table 1. In the Figure 3, we then plots the per-axis effect on R@5 and N@5. From the ablation, we notice a few pattern as below:

**Smaller models do better on this dataset.** The default $d_{\text{model}}$=384 is a bit over-parameterised for ∼12K items as cutting $d$ to 192 able to beats the default on every metric and the invalid-generation rate drops to 0.1%. In the other side, using $d$=768 seems to overfits and had detrimental effect, this also pushes invalid generations up to 11%. The 2-layer model also beats the default and trains ∼2× faster.

$L$=4 **helps a little,** $L$=2 **hurts.** More residual levels give richer hierarchy. With $L$=2 only ∼ 76% of items have a unique prefix, which limits discrimination. $L$=4 able to beats the default on all four metrics.

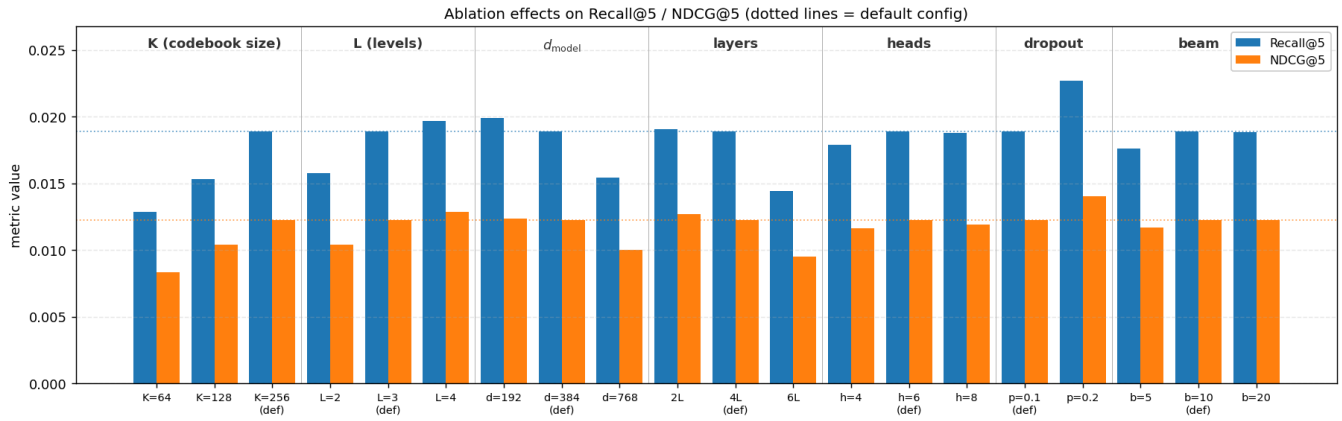


**Figure 3: Per-axis effect of each ablation on Recall@5 (blue) and NDCG@5 (orange). Dotted horizontal lines mark the default-config metrics.**

**Smaller codebooks lose recall.** $K$=64 and $K$=128 both lose around 1.2–1.5× on R@5 because they compress harder and lose item-level distinguishability.

**Dropout has the biggest effect.** Changing the dropout value from 0.1 to 0.2 gives +20% on R@5 and +16% on N@10, without any cost to invalid generations or training time. This was the clearest single fix for the train/val gap visible in Figure 2.

**Beam size matters less than expected.** $b$=10 and $b$=20 are nearly identical on R@10 and N@10. $b$=5 trivially gives R@10 = R@5 (the beam only contains 5 candidates).

## 3.4 Cold-start

For the cold start experiment, we held out a random 5% of items which are 596 of 11,924 as cold. RQ-VAE is still trained on all items as it only sees content embeddings, so each cold item still receives a Semantic ID, but cold items are stripped from every user's training history. Test users are then split into warm-test (18,479, normal test target) and cold-test (933, target is cold). We used $L$=4 for this experiment rather than $L$=3 + suffix so cold items each have a unique suffix value. The Transformer has never seen during training, so any $L$=3+suffix run would give zeros by construction. With $L$=4 ($256^4 \approx 4.3$ B tuples) cold items share content-similar tokens with warm items at every position.

| Test split | users | R@5 | R@10 | N@5 | N@10 |
|---|---|---|---|---|---|
| Warm-test | 18,479 | 0.0202 | 0.0310 | 0.0131 | 0.0166 |
| Cold-test | 933 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |

**Table 2: Cold-start results with $L$=4 Semantic IDs. Warm-test metrics line up with the default config. Cold-test metrics are zero across the board.**

The warm-test row is roughly in line with our default-config numbers (∼0.020/0.031/0.013/0.017), so the cold-filtered training did not hurt the warm setting much. The cold row is zero on every metric.

The reason, as far as we can tell, is that every individual codebook token a cold item uses also appears in some warm item (because the codebook is 100% used), but the specific 4-token combination that identifies a cold item never appears in any training sequence. The Transformer's autoregressive distribution $p(c_l \mid c_{<l}, \text{history})$ therefore puts essentially no probability on the cold item's prefix continuations and beam search never proposes them. The cold-start property TIGER demonstrates in the paper relies on much denser tuple coverage (millions of items) and a larger content encoder, neither of which we have here. We discuss possible fixes in Section 4.

## 3.5 Qualitative analysis

To check that prefix-sharing items are semantically similar, we sampled ten Semantic-ID prefix groups (first $L-1$=2 codebook tokens) that have at least 4 members and inspected their titles. Most groups map to a clean product category (Table 3): Fisher-Price infant musical toys, Hot Wheels / Tonka / Cars die-cast vehicles, Syma RC helicopters, Star Wars X-Wing miniatures, Crayola / Play-Doh art supplies, Barbie and Madame Alexander dolls. This is consistent with the 100% codebook usage we saw in Section 3.1: the RQ-VAE has learned content clusters that line up with how a person would group the items.

# 4 Discussion

Our default-config Transformer reaches about a third of the test scores the TIGER paper reports on Toys & Games. The main reasons we see are (a) a single Colab T4 budget that limits one training run to about 77 minutes, (b) the base-size Sentence-T5 content encoder instead of a larger one and (c) some signs of overfitting that the default hyperparameters do not fully control: the train/val gap in Figure 2 is visible and the dropout-0.2 ablation fixes a lot of it (+20% on R@5). The rest of the ablation grid points in the same direction which are the default config is slightly too big for a ∼ 12K-item catalogue, with smaller models ($d$=192, 2 layers) beating the default and the $d$=768 variant emitting 11% invalid token sequences.

The qualitative analysis lines up with what we expected: prefix-shared items belong to the same brand or product family and the

| Prefix | Sample items (titles abbreviated) |
|---|---|
| (0, 154) | Fisher-Price Sit-To-Stand Giraffe; Roll-Along Snail / Turtle; Smart Bounce & Spin Pony |
| (2, 146) | Sunshine Symphony; VTech Crawl Ball; Giggle Sound Station; Big Keyboard Fun; Fisher-Price Poppity-Pop Dino |
| (2, 170) | Fisher-Price Laugh & Learn Birdbath / Peek-a-Boo Cuckoo / Singin' Soccer Ball / Sweet Sounds Picnic |
| (10, 134) | Syma S107 / S107G mini RC helicopters (various colours / channels) |
| (65, 5) | Hot Wheels Starter Set; Tonka Retro Bulldozer; Disney/Pixar Cars Rip-Start; Cars Off-Road Rally Trackset |
| (89, 69) | Star Wars X-Wing Miniatures core set + X-Wing / Y-Wing / TIE Fighter / Dice expansions |
| (99, 78) | Crayola Colored Pencils; Play-Doh Rollers & Cutters; Lalaloopsy Mini Sketch Book; Crayola Glitter Pad |
| (228, 215) | Barbie Ball Gown; doll hangers for 18" dolls; Madame Alexander Friends; Talkin' Barbie Doll |

**Table 3: Sample of Semantic-ID prefix groups and their items. Each prefix corresponds to a coherent product category.**

codebook saturates at 100% usage on every level, with only 4.30% prefix collisions on ∼12K items. So the Semantic IDs themselves look healthy, the Transformer that is the bottleneck.

The cold-start experiment is the one place where we got a clearly negative result. Even with $L$=4 (which avoids the suffix-token trap), cold items get exactly zero retrieval. Every individual codebook token they use appears in the training set, but the specific 4-token combination does not and beam search will not propose token sequences the model assigns near-zero probability. The cold-start property TIGER demonstrates in the paper relies on much denser tuple coverage and a stronger content encoder. We did not have either available, so the result on our scale is honest but unimpressive.

*Issues we ran into.* A few things we were not expecting and cost us some debugging time. (i) Updating codebooks with the gradient of the codebook MSE loss collapses the deeper levels very quickly, the residual signal shrinks after each level so the gradients vanish. Switching to EMA codebook updates with dead-code reset fixed this and pushed assignment-level usage to 100% at every level. (ii) If early stopping only watches val recon, it can save an undertrained checkpoint because recon plateaus well before codebook usage stabilises as we gated stopping on both. (iii) fp16 training at peak lr $10^{-3}$ gave occasional loss explosions around epoch 10. Lowering peak lr to $3{\times}10^{-4}$, tightening grad clipping to 0.5 and skipping optimiser steps on non-finite gradients fixed it. (iv) The invalid-generation fraction turned out to be a very fast diagnostic for over-parameterisation, the $d$=768 run's 11% invalid fraction was a clear warning sign before we even looked at the recall numbers.

*Things we would try next.* The cleanest follow-ups, in roughly increasing cost, would be: (1) promote dropout 0.2 to the new default as it was the biggest single win in our grid. (2) add label smoothing to the cross-entropy loss to push back further against the remaining train/val gap. (3) use constrained beam search that only generates tokens from the correct codebook range at each decoding position, which would zero out invalid generations and recover the 11% wasted by the $d$=768 configuration. (4) for cold-start specifically, give the Transformer an auxiliary content-embedding input (added to the token embedding for each history item) so the model has a non-zero prior over cold items it has never seen as token tuples. (5) a longer training budget and a larger sentence encoder (`sentence-t5-large` or `xl`) to close more of the gap to the paper's reported numbers.

## 5 Conclusion

We reimplemented the TIGER pipeline end-to-end and trained it on Toys & Games on a single Colab T4. The RQ-VAE half worked the way we hoped which are codebooks saturated, prefix neighbourhoods looked like real product categories and only a small fraction of items needed a disambiguation suffix. The Transformer half is where most of the gap to the paper's reported scores came from and the clearest take-away from the ablation grid is that the default configuration is slightly over-sized for a ∼12K-item catalogue modest changes (more dropout, smaller $d_{\text{model}}$, fewer layers) all moved the metrics in the right direction. Cold-start did not work at our scale, which we read more as a coverage problem than a problem with the recipe itself. So overall, the generative-retrieval idea is reproducible on a small dataset and a modest compute budget, but how far the metrics climb depends a lot on things outside the architecture like dataset size, the strength of the content encoder and how much training time given.

## References

[Kang and McAuley(2018)] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *IEEE International Conference on Data Mining (ICDM)*. https://arxiv.org/abs/1808.09781

[McAuley et al.(2015)] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-based Recommendations on Styles and Substitutes. In *ACM SIGIR Conference on Research and Development in Information Retrieval*. https://jmcauley.ucsd.edu/data/amazon/

[Ni et al.(2022)] Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith Hall, Daniel Cer, and Yinfei Yang. 2022. Sentence-T5: Scalable Sentence Encoders from Pre-trained Text-to-Text Models. In *Findings of the Association for Computational Linguistics (ACL)*. https://arxiv.org/abs/2108.08877

[Rajput et al.(2023)] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, et al. 2023. Recommender Systems with Generative Retrieval. In *Advances in Neural Information Processing Systems*, Vol. 36. https://arxiv.org/abs/2305.05065

[van den Oord et al.(2017)] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Neural Discrete Representation Learning. In *Advances in Neural Information Processing Systems*. https://arxiv.org/abs/1711.00937