TIGER

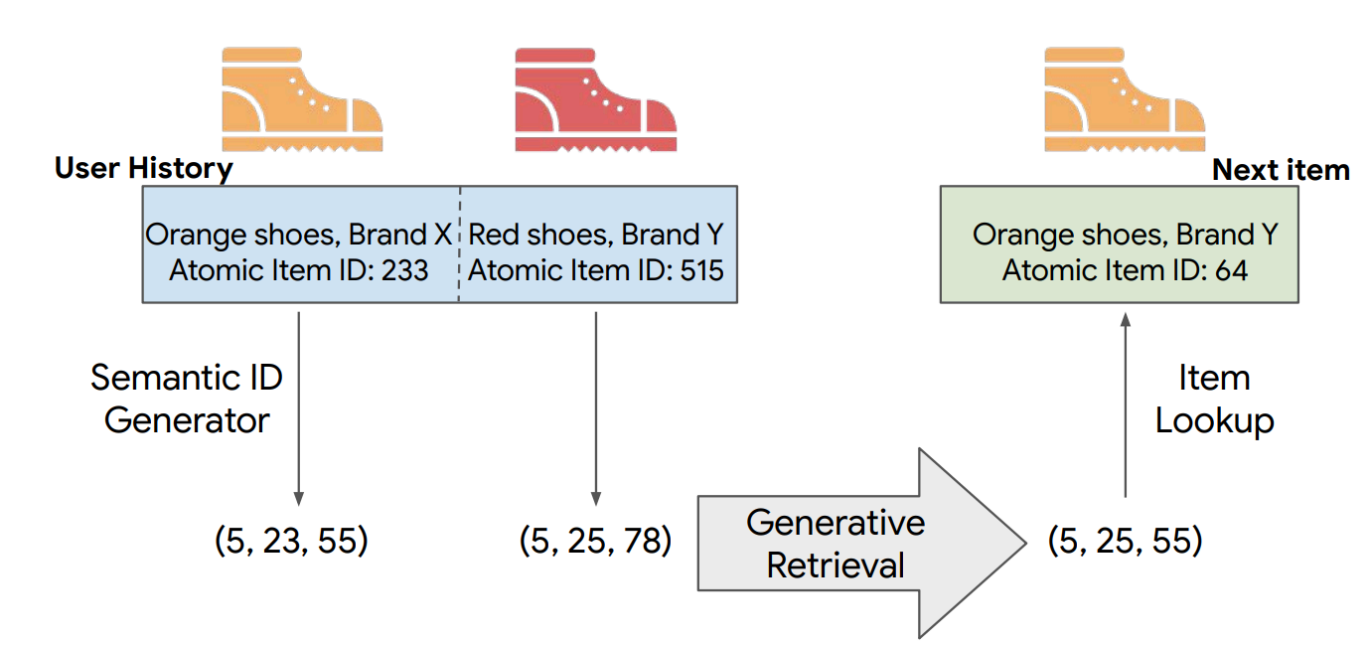
> *文章链接：https://arxiv.org/pdf/2305.05065*
>
> *机构：Google*
>
> *发布时间：2023*
>
>

在传统推荐系统中，物品通常只使用孤立的、无语义的ID来表示，但这样缺乏物品本身的语义信息，使得模型难以直接理解物品间的内在联系（例如两个不同型号的运动鞋本质上高度相似），只能通过用户历史序列中海量的共现数据间接学习，效率低下。另外，在冷启动中模型难以推荐训练集中没出现过的物品，因为学习时词汇表里根本就没有这些新物品的ID。

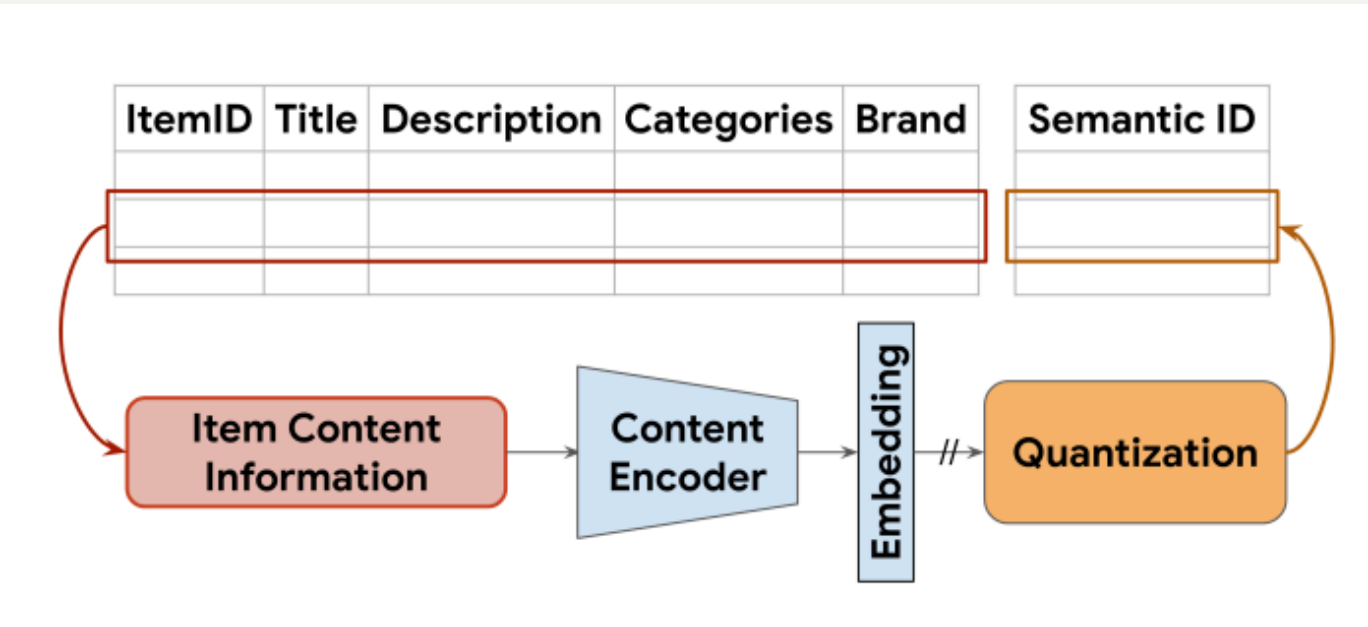
TIGER（Transformer Index for GEnerative Recommenders, 2023）试图**改进物品的表示方式**，其将每个物品都表示为由一组code（码字）表示的**Semantic ID（SID）**，从而能够涵盖物品内在的语义信息。这样即使某新商品未在训练中见过，也能够基于其语义特征进行合理的推荐。通过引入语义ID，模型不仅能够在相似物品间共享知识，还能用更紧凑的方式表示庞大的物品库。用户历史交互商品生成的码字序列可以进一步输入到天然适应离散输入的transformer中，然后自回归地直接预测下一个物品的SID，然后通过查表来找到其对应的物品即可。

这打破了双塔召回中使用user query向量通过ANN检索最相似的物品向量的范式，**实现了在推理时直接端到端预测下一个召回目标**。本质上，TIGER将transformer参数本身当成了用于召回的索引，而不是显式地构建一个ANN索引来储存所有物品向量。

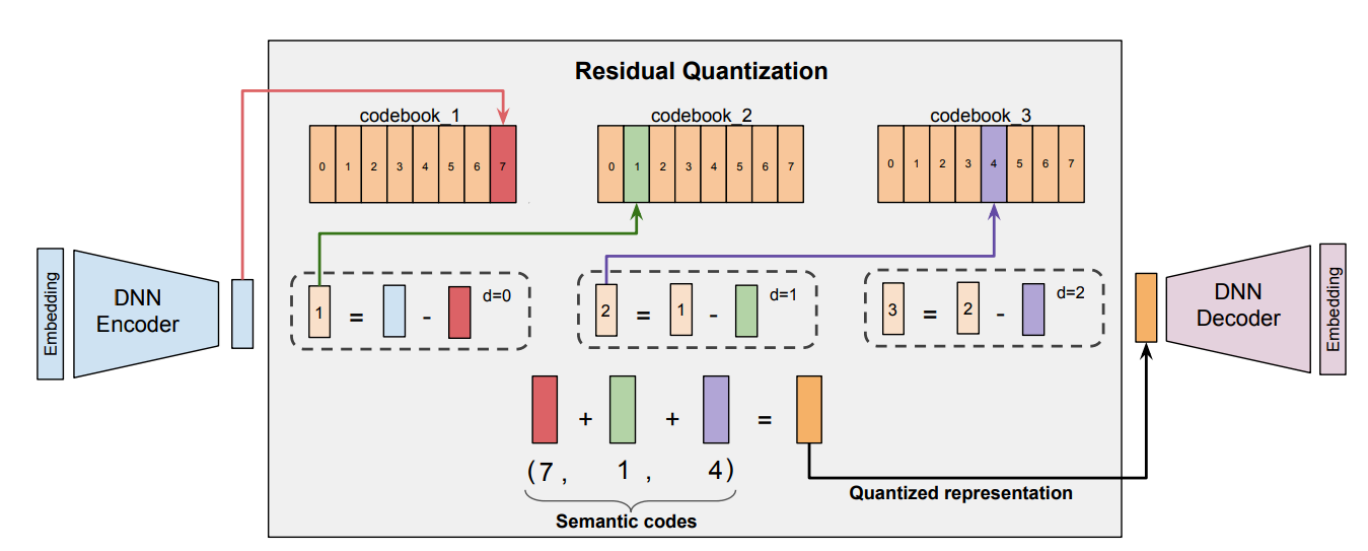
例如下图中，用户历史中的两个鞋子的SID分别为`(5, 23, 55)`,`(5, 25, 78)`，将它们拼起来`(5,23,55,5,25,78)`输入到transformer后，依次预测输出`5,25,55`，也即预测下一个商品的SID为`(5,25,55)`，查表可得其对应ID为64的那个商品，则可以将其返回作为下一个推荐结果。



TIGER的第一步是为商品item构建SID。具体而言，其首先使用一个预训练好的文本encoder（例如SentenceT5）来将商品的文本特征（ItemID+Title+Description+Categories+Brand）转换一个为稠密的embedding向量$\mathbf{x}$，然后将其输入**RQ-VAE**来构建SID：



相比于VQ-VAE只使用一个code来表示输入在latent space中的信息，RQ-VAE通过逐层残差量化来为输入生成一系列细粒度、层次化的code，使得一个输入在latent space中的信息被编码为一组多个层次化的code，从而在不需要维护一个巨大codebook的情况下，指数级扩展表示能力（每一层提供一个code，多层的code排列组合可以形成极其丰富的表示，同时每层只需维护一个不算太大的codebook）。



具体而言，输入向量经过RQ-VAE的Encoder后，得到其latent表示向量$\mathbf{z}$，并初始化第0层的残差为$\mathbf{r}_0 = \mathbf{z}$。设模型中共有$m$个层级，每个层级$d \in \{0, 1, \cdots, m-1\}$都各自有一个独立的codebook $\mathcal{C}_d$。在第d层时，通过在其codebook中寻找与当前输入残差$\mathbf{r}_d$最近的code来完成量化，从而得到该层提供的code id c_d：

$$c_d = \arg\min_k ||\mathbf{r}_d - \mathbf{e}_k||^2, \quad \mathbf{e}_k \in \mathcal{C}_d$$

然后计算该层的残差，也即当前残差$\mathbf{r}_d$和其最近的code $\mathbf{e}_{c_d}$之间的差距，并将其作为下一层的输入残差：

$$\mathbf{r}_{d+1} = \mathbf{r}_d - \mathbf{e}_{c_d}$$

如此依次在各层重复该过程，并采集每一层选取的code c_d，最终得到由m个code组成的SID元组：$(c_0, c_1, \cdots, c_{m-1})$。例如在上图示例中，选中的code依次为7,1,4，因此该输入的SID就是$(7, 1, 4)$

> *也即，第一层是寻找和输入latent表征最近的code，第二层是寻找和第一层残差最近的code，第三层是寻找离第二层的残差（第一层的残差的残差）最近的code.....*
>
> *实践中输入embedding维数为768，RQ-VAE中选取了3层codebook，每层的codebook大小都是256，每个code的维数都是32。*

在训练RQ-VAE时，得到所有层的code后，将这些code做累加融合成一个单一向量 $\hat{\mathbf{z}} = \sum_{d=0}^{m-1} \mathbf{e}_{c_i}$，然后将其输入RQ-VAE decoder中试图重建原始输入$\mathbf{x}$。其loss函数和VQ-VAE基本一致，也是由重建损失和量化损失构成，其中量化损失为所有中间层的量化损失之和：

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_{recon} + \mathcal{L}_{rqvae} \\ &= ||\mathbf{x} - \hat{\mathbf{x}}||^2 + \sum_{d=0}^{m-1}(||sg[\mathbf{r}_i] - \mathbf{e}_{c_i}||^2 + \beta||\mathbf{r}_i - sg[\mathbf{e}_{c_i}]||^2)\end{aligned}$$

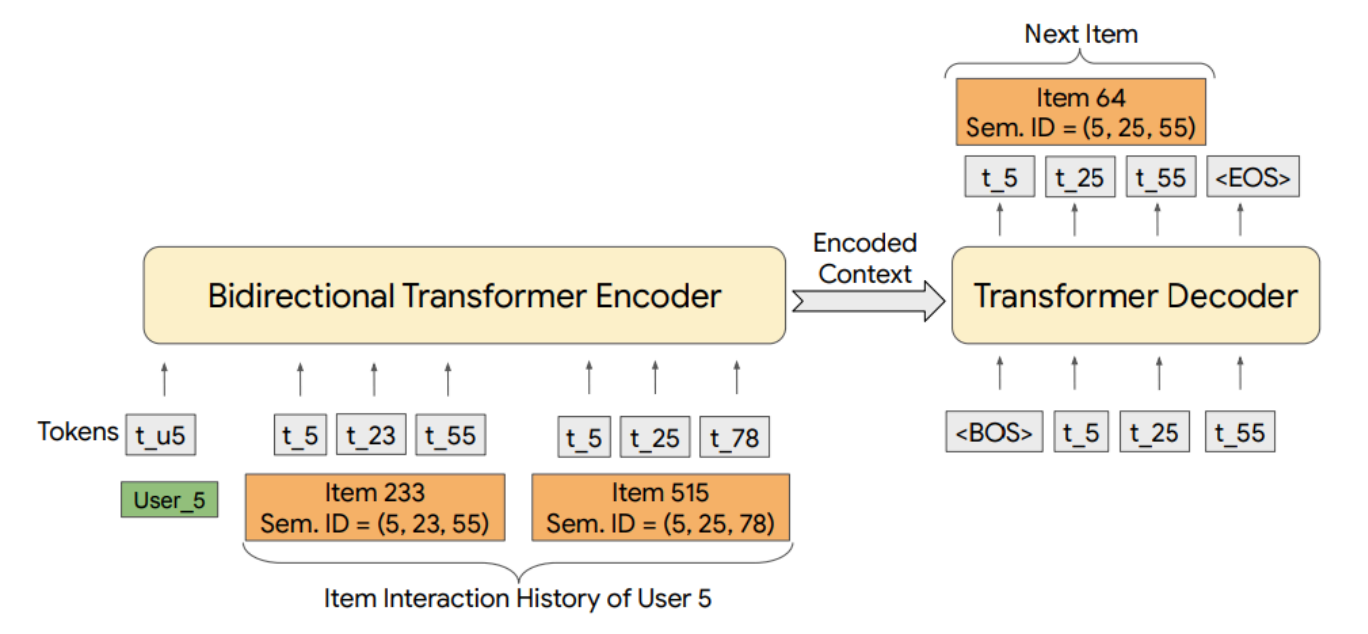
训好后的模型encoder+quantizer即可为每个物品生成SID，然后构造item-SID和SID-item查找表，以便推理时将预测出来的下一个SID映射回其对应的物品。

在推理时，需要的只是输入物品的SID元组$(c_0, c_1, \cdots, c_{m-1})$，训练中后续的累加和decoder重建步骤均丢弃。得到输入序列中的n物品的SID后，将它们连接起来形成一个长序列作为transformer的输入（实际上输入的是这些SID对应的那些code向量）：

$$\begin{gathered}((c_{1,0}, \ldots, c_{1,m-1}), (c_{2,0}, \ldots, c_{2,m-1}), \ldots, (c_{n,0}, \ldots, c_{n,m-1})) \\ \Downarrow \\ (c_{1,0}, \ldots, c_{1,m-1}, c_{2,0}, \ldots, c_{2,m-1}, \ldots, c_{n,0}, \ldots, c_{n,m-1})\end{gathered}$$

预测目标就是预测下一个item的SID：$(c_{n+1,0}, \cdots, c_{n+1,m-1})$

预测出来SID后，通过查找表来找到对应的item返回，作为召回结果。



其他细节：

- 有时多个物品可能产生相同的SID，为了避免冲突会在SID末尾追加一个额外的索引来区分它们，如$(12, 24, 52, 0)$和$(12, 24, 52, 1)$
- 为了避免码本坍塌（绝大多数输入被映射成相同的几个SID，使得码本使用率极不均衡），对codebook使用基于KMeans的初始化。

  具体而言，对于训练时的第一批数据的latent向量$\{\mathbf{z}(\mathbf{x}_i)\}_i$，对它们执行聚类数量为codebook大小$K$的KMeans，并得到$K$个聚类中心，用这些聚类中心当作第一层codebook中每个code的初始值。第二层的codebook初始化也即使用第一层的残差（实际上也即latent向量和第一层最近聚类中心的距离）再做KMeans，依次类推初始化所有层。这样可以让codebook在训练一开始就已经覆盖了latent space的分布特点，使得其能有效覆盖空间，不会因为随机初始化导致很多code因为用不到而collapse。
- 如果下一个生成的SID未对应查找表中任何一个物品，则可以使用某些策略找到最接近的item
- 除了SID tokens之外，还设置了2000个user-specific tokens，使用hashing trick将原始user id映射到这2000个之一，并将这个user token放在输入的SID序列之前，从而提高模型的个性化推荐能力。