清华大学学报（自然科学版）
J Tsinghua Univ (Sci & Technol)

Semantic ID 的编码方式对比

算法小小怪下士

2025 年 12 月 19 日

摘要

在生成式推荐与检索中，传统的 ID 面临着词表基数过大、缺乏语义信息以及冷启动困难等挑战，难以直接应用于基于 Next Token Prediction 的生成范式。Semantic ID 通过将物品表征为离散的 Token 序列，不仅赋予了 ID 语义特性，还能有效协同 LLM 的推理能力，成为解决上述问题的关键方案。

本文档系统性地梳理了主流的 Semantic ID 编码方式，包括 VQ-VAE, RQ-VAE, Res-Kmeans, FSQ, RQ-OPQ 以及 OPMQ，并详细对比了各方法的数学原理与优劣势。进一步地，本文深入剖析了 Semantic ID 生成过程中的一系列问题，如何评估 codebook 的质量，如何初始化 Codebook 等。

1 各方法的公式与原理

1.1 VQ-VAE (Vector Quantized VAE)

VQ-VAE 是最基础的离散化编码方式，将连续的 Embedding 映射到最近的 Codebook 向量。

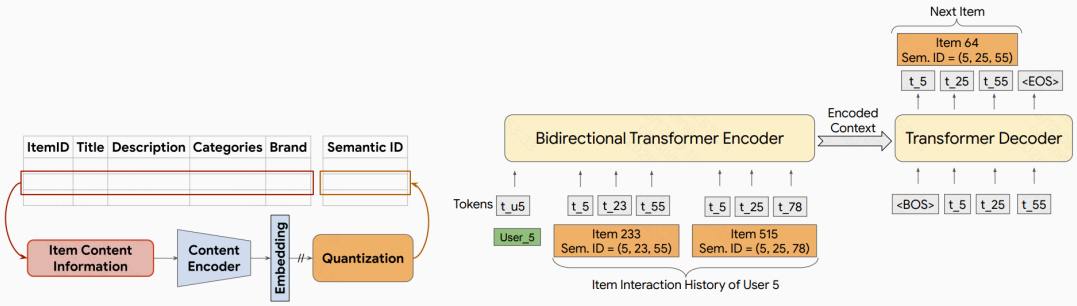
- **核心机制**：单层量化。
- **Loss**：包含 Reconstruction Loss、Codebook 损失和 Commitment 损失。
- **Reconstruction Loss**: 目标是确保解码出来的结果尽可能还原原始输入，作用对象是 Encoder 和 Decoder。
- **Codebook Loss**: 目标是更新 Codebook 的向量，使其靠近编码器的输出，作用对象是 Codebook，可以把这一步理解为 K-means 聚类中的更新聚类中心，强迫 codebook 中的向量移动到数据分布密集的区域，从而更好地代表数据。
- **Commitment Loss**: 目标是约束 Encoder 的输出不要偏离码本向量太远，防止编码器的输出在 Latent Space 中随意跳动，即便只发生微小变化，也可能映射到完全不同的 codebook 向量，作用对象是 Encoder。

$$\begin{aligned}\mathcal{L}_{VQ} = & \underbrace{\|\boldsymbol{x}-\hat{\boldsymbol{x}}\|^2}_{\text{Reconstruction Loss}} \\ & + \underbrace{\|sg[\boldsymbol{z}_e(\boldsymbol{x})]-\boldsymbol{e}\|^2}_{\text{Codebook Loss}} \\ & + \beta \underbrace{\|\boldsymbol{z}_e(\boldsymbol{x})-sg[\boldsymbol{e}]\|^2}_{\text{Commitment Loss}}\end{aligned} \tag{1}$$

其中，$sg[\cdot]$ 表示 stop-gradient 操作，$\boldsymbol{z}_e(\boldsymbol{x})$ 是编码器输出，$\boldsymbol{e}$ 是最近邻的码本向量，β 通常取为 0.25。

1.2 RQ-VAE (Residual-Quantized VAE)

- **核心机制**：递归地对上一层的残差进行量化。$r_0 = \boldsymbol{z}, r_d = r_{d-1} - \boldsymbol{e}_{c_{d-1}}$，相比于 VQ-VAE 能实现更低的 reconstruction loss。
- **Loss 公式**：重构损失加上每一层的量化损失。



(a) Semantic ID generation for items using quantization of content embeddings.

(b) Transformer based encoder-decoder setup for building the sequence-to-sequence model used for generative retrieval.

图 1 Overview of RQ-VAE

Algorithm 1: Balanced K-means Clustering

Input: 物品集合 $\mathcal{V}$，聚类簇数量 K
Output: 优化后的码本 $C_l = \{c_1^l, \ldots, c_K^l\}$

计算每个簇的标准大小 $w \leftarrow |\mathcal{V}|/K$;
随机选择初始化聚类中心 $C_l = \{c_1^l, \ldots, c_K^l\}$;
repeat
  初始化未分配集合 $\mathcal{U} \leftarrow \mathcal{V}$;
  for 每个簇 $k \in \{1, \ldots, K\}$ **do**
    根据与中心 c_k^l 的距离对 $\mathcal{U}$ 进行升序排序;
    分配前 w 个样本 $\mathcal{V}_k \leftarrow \mathcal{U}[0 : w-1]$;
    更新聚类中心 $c_k^l \leftarrow \frac{1}{w} \sum_{r^l \in \mathcal{V}_k} r^l$;
    移除已分配的物品 $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{V}_k$;
until 分配结果收敛 *(Assignment convergence)*;
return 优化后的码本 $C_l = \{c_1^l, \ldots, c_K^l\}$

$$\mathcal{L}_{RQ} = \mathcal{L}_{recon} + \sum_{d=0}^{m-1} \left(\|sg[\boldsymbol{r}_d] - \boldsymbol{e}_{c_d}\|^2 + \beta \|\boldsymbol{r}_d - sg[\boldsymbol{e}_{c_d}]\|^2 \right) \tag{2}$$

其中 m 是量化的层数，$\boldsymbol{e}_{c_d}$ 是第 d 层选中的码本向量。

1.3 Balanced-Res-Kmeans

OneRec 指出 RQ-VAE 容易出现沙漏现象 [6]，即物品的 SID 会集中在中间层的某个向量，因此使用基于 K-Means 的残差聚类, 在划分时强制每个簇包含相同数量的样本（$|V|/K$），以最大化熵。详细的算法介绍可以见下面的 Algorithm 1.

- **核心机制**：非梯度下降训练 codebook，而是通过迭代优化聚类中心。
- **优化目标**：最小化每一层的残差向量与聚类中心的欧氏距离。仅有 reconstruction loss。

$$\boldsymbol{s}_i^l = \arg\min_k \|\boldsymbol{r}_i^l - \boldsymbol{c}_k^l\|_2^2, \quad \text{where} \quad \boldsymbol{r}_i^{l+1} = \boldsymbol{r}_i^l - \boldsymbol{c}_{\boldsymbol{s}_i^l}^l \tag{3}$$

1.4 FSQ

FSQ [4] 的核心思想是在生成 Semantic ID 的时候，不再显式的码本查找（Codebook Lookup）。传统的 RQ-VAE 需要训练一个额外的 Codebook，并在表中寻找最近邻。而 FSQ 直接将向量的每一

个维度映射到一个有限的整数集合中，这些整数的组合直接构成了 Token 的索引。推理阶段使用的 Lookup Table，类似于 LSH 算法，是训练 Transformer 而“全新初始化”并“重新学习”出来的。

FSQ 的计算过程非常直接，主要包含投影、归一化、缩放、取整四个步骤：给定一个输入子向量 $\boldsymbol{e}_i^k$（Item Embedding 的一部分），FSQ 的计算公式如下

$$FSQ(\boldsymbol{e}_i^k) = R\left[(L-1)\cdot\sigma(\mathcal{T}_{in}(\boldsymbol{e}_i^k))\right] \tag{4}$$

其中各个组件的含义为：$\mathcal{T}_{in}$ (投影)：一个仿射变换（线性层 $W\boldsymbol{x}+b$），将输入向量的维度映射到 d_{fsq} 维。

σ (归一化)：Sigmoid 函数，将数值压缩到 $(0,1)$ 区间。

L 是量化的层级数（即每个维度有多少种可能的取值）

$R[\cdot]$ (取整)：Rounding 函数，将数值四舍五入为最近的整数

结果：输出是一个 d_{fsq} 维的整数向量，每个维度的值都在 0 到 $L-1$ 之间。这个整数向量本身就是离散的 token 表示。FSQ 和下面的 OPQ 优点是尽可能多地对有效信息进行 token 化，缺点是这两种方法未能以层级化的方式表征相似物品之间的核心属性。

1.5 RQ-OPQ

OneSearch [1] 指出单纯的 RQ-Kmeans 会丢失 Item 的 Unique features。例如“电子产品 -> 手机 -> 苹果手机”。此时，所有的“苹果手机”可能都被映射到了同一个 Semantic ID 前缀。这导致了信息的损失，无法区分具体的“红色、256G、Pro 版”等细微差别。因此，Onesearch 采用 RQ 编码层级语义，末端残差使用 OPQ 编码 (Optimized Product Quantization)。

OPQ/PQ：将高维向量切分为多个低维子空间。每个子空间单独使用一个小的码本。结果：最终的表达能力是所有子空间码本大小的乘积。这种组合能力使得 OPQ 能够以极小的存储代价，表达海量的、极其细微的特征差异，从而保留了物品的特有信息，提高了独立编码率 (ICR) 。

普通的 PQ 知识简单的把向量切开，但如果原始特征之间的各个维度之间存在相关性（例如“长”和“宽”通常是相关的），简单切分会破坏这些结构信息，导致量化误差大。OPQ 的解法：在切分之前，OPQ 会学习一个正交旋转矩阵 R。

目标：$\min_{R,C}\sum\|\boldsymbol{x}-\text{quantize}(R\boldsymbol{x})\|^2$。这个旋转操作会将数据旋转到最适合切分的方向（类似于 PCA 主成分对齐），使得各个子空间之间的数据尽可能独立。

- **核心机制**：RQ 处理前几层（Hierarchy）+ OPQ 处理末层残差（Distinction）。
- **优化目标**：前 L 层同 Res-Kmeans，最后一层优化正交旋转矩阵 R 和码本 C。

$$\min_{R,C}\sum_i\|\boldsymbol{r}_i^{last}-\text{quantize}(R\boldsymbol{r}_i^{last})\|^2 \tag{5}$$

1.6 OPMQ (Orthogonal Parallel Multi-expert Quantization)

STORE [10] 论文提出了一种并行的量化方案，解决了深层残差量化的效率问题。

- **核心机制**：使用 K 个并行专家网络，每个专家在不同的正交子空间进行编码。
- **Loss 公式**：包含重构损失和正交正则化损失。

$$\mathcal{L}_{OPMQ} = \mathcal{L}_{recon} + \lambda\mathcal{L}_{orth} \tag{6}$$

$$\mathcal{L}_{recon} = \|\boldsymbol{e}_p - \text{Decoder}\left[\sum_{i=1}^{K}(\boldsymbol{z}_i + sg(\boldsymbol{s}_i-\boldsymbol{z}_i))\right]\|^2 \tag{7}$$

$$\mathcal{L}_{orth} = \|VV^T - I\|_F^2 \tag{8}$$

其中 V 是所有专家的参数矩阵集合。

表 1 不同 Semantic ID 编码方式对比

方法	核心特征	优势 (Pros)	劣势 (Cons)
VQ-VAE	单层量化	结构简单，实现容易。	缺乏层级语义，码本利用率低，难以表达细粒度特征。
RQ-VAE	多层残差 (串行)	建立层级语义，适合 Beam Search。	存在“沙漏现象”，导致码本坍塌 (Codebook Collapse)，长尾 Item 区分度低。
Res-Kmeans	迭代聚类 (非梯度)	几何语义更强；配合 Balanced 策略可最大化码本利用率。	虽然解决了部分坍塌，但对极度细粒度的 Unique features 表达仍有限。
FSQ	隐式码本 + 标量量化	**无码本坍塌**；结构极简（无需维护聚类中心表），直接通过数学公式计算 ID，词表利用率极高。	取整操作不可导（需 STE）；语义层级性不如 RQ 强，更像是一种高维网格坐标。
RQ-OPQ	残差 + 乘积量化	结合了 RQ 的层级性和 OPQ 的高区分度，显著提升 ICR。	编码流程相对复杂，需结合两种算法；主要用于解决电商搜索中的相关性约束问题。
OPMQ	并行 + 正交	**并行高效**；利用正交约束避免特征冗余，解决 Interaction-Collapse。	需要预训练 Embedding；不再是纯粹的“残差”逻辑，而是“多视角”子空间逻辑。

2 方法对比总结

表 1 展示了不同编码方式的详细对比。

3 评估 codebook 质量的维度

3.1 Codebook utilization/coverage

codebook utilization 的定义是指实际被激活（即至少被分配给一个物品）的码本向量数量占码本总大小的比例。

$$\text{Utilization} = \frac{\text{Non-empty codebooks}}{\text{Total codebooks}}$$

这就引出了另一个概念，码本坍缩（Codebook Collapse）是指在在训练过程中，Codebook 中的 utilization 极低，导致有效码本容量急剧下降的现象。

我们首先需要区分两个概念：死码本 (Dead Codes)：指从未被 Encoder 输出选中的 Codebook 向量。它们在训练过程中无法获得梯度更新，永久停滞在初始化状态。码本坍缩是一个更广泛的概念，指 Codebook 的有效分布退化。即便是所有 Code 都被使用了，如果分布极度不均匀（例如 1% 的 Code 承载了 99% 的数据），这也是一种“坍缩”。

然后，我们是否需要解决码本坍缩？这个问题的本质矛盾是 Zipf 定律与最大熵原理的博弈。自然界的 Zipf 分布：自然界的数据信号（如音频中的静音与基频、文本中的虚词、电商中的热门商品）本质上是分形的、长尾的。极少数的模式占据了绝大多数的时间或频率。如果顺从数据的自然天性，Codebook 的使用率必然服从 Zipf 分布（幂律分布）。

然而，在离散化表示学习中，无论是音频还是推荐系统，都面临着相同的物理约束：定长的帧率、固定的词表大小、固定的信道带宽。从信息论的角度来看，为了在有限的词表空间内传递最大的信息量，因为均匀分布具有最大熵（$\log K$），所以我们希望 Codebook 的分布能够逼近均匀分布 (Uniform Distribution)。

如果顺从自然的 Zipf 分布，定长的 Codebook 将面临严重的利用率坍缩，大量长尾的 Code 极少被使用，造成了带宽和参数的浪费。为了解决这个问题，现有的工作有的强制每个 Cluster 的样本数

量相等，或在 Loss 中加入熵正则项。这种强行将“长尾分布”拉伸为“均匀分布”的操作，一方面可能造成，语义稀释（Semantic Fragmentation）：模型被迫将原本极其常见的模式强行拆分、散射到多个不同的 Code 中，以填充频率的空缺。另一方面造成的后果是语义聚合（Semantic Aggregation）：罕见的模式被强行聚类到同一个 Code 中。这导致了单个 Token/Code 的语义变得模糊甚至无意义。语义不再由单一的原子 ID 承载，而是必须通过 Token 的组合才能涌现出来。

如何说明一个模型是否真的需要解决坍缩呢？目前最直接的方法就是实验。对比“自然坍缩模型”与“强制均匀模型”。如果强制均匀显著提升了下游任务（如 OneRec 所示），则证明为了最大化信息熵而牺牲部分自然语义是值得的。

如何解决熵坍缩呢？除了后面说的解决 collision 的方法可以试一下，还可以使用 Multi-codebook 的方式，比如小红书的 PQ-VAE，中科大的 AlphaFuse。

3.2 Collision

在快手最新发布的综述中，论述了这部分的工作的做法，可供参考。Collision 的定义是：多个相似商品对应同一个 SID 的情况。这种冲突在物品定位，即 item grounding 时引入了歧义，因为 SID 序列不再能唯一地标识单个物品，从而使得额外的消歧策略成为必要。因此，高冲突率会导致生成式推荐模型的性能显著下降。冲突源于量化方法的固有局限性：在 RQ-VAE 或 ResKmeans 中，学习到的聚类中心往往分布不均或发生坍缩，大多数物品聚集在少数几个主导的聚类中心周围。

为了缓解冲突问题，最近的工作在码本训练期间引入了额外的优化目标，以鼓励更均衡的聚类中心分布。SaviorRec [11] 在量化过程中结合了 Sinkhorn 算法，采用熵正则化损失来强制物品更均匀地分配给聚类中心。OneRec 和 LETTER [8] 利用针对每个残差的受限 K-means，限制了分配给每个聚类中心的物品的最大数量。

除了训练时的去冲突策略外，另一类工作通过在最后一层添加额外的 Token 位置来缓解冲突。TIGER [7] 在 SID 的末尾添加一个随机 Token，而 CAR [9] 则添加物品的 Sparse ID。此外，OneSearch 使用 OPQ 在最终的 SID Token 中编码独特特征，从而提高了 SID 的区分度。

FORGE 采取了 KNN 的策略。对于一个物品，不只生成唯一的最佳 SID，而是采样出多个候选 SID。按照与物品特征的相似度得分对这些候选 SID 进行排序，依次检查每个候选 SID 当前已分配的物品数量，一旦找到一个关联物品数少于预设阈值的 SID，就将该物品分配给它。传统观念认为 ID 必须唯一。但 FORGE 的实验表明，使用 Random 策略强制打散最后一层 ID（牺牲语义），效果反而优于基于语义的 KNN。这是否说明，在深层残差中，“让每个 ID 都有机会被训练”比“这个 ID 代表什么”更重要？

3.3 Token distribution entropy [12]

该指标用于评估 Tokenizer 生成的 Semantic ID 中，Codebook 的使用均匀程度。基于 Shannon Entropy，Codebook 的大小为 N（例如 8192），p_i 表示第 i 个 token 被使用的频率：

$$H = -\sum_{i=1}^{N} p_i \log p_i$$

统计所有生成的 Semantic ID 中，Codebook 里每个 token v_i 出现的次数。计算每个 token 的出现概率 $p_i = \frac{\text{count}(v_i)}{\text{Total Tokens}}$。代入上述公式计算熵值。

含义：数值越大越好。熵越大，代表 Token 的分布越接近均匀分布（Uniform Distribution）。这意味着 Codebook 中的每个向量都被充分利用了，没有出现“某些 Token 被过度使用，而其他 Token 闲置”的情况（即 Codebook Collapse 现象）。

3.4 Gini Coefficient [3]

衡量 SID 分布的公平性（Fairness）或不平衡程度。首先统计每个 SID 对应的物品数量 I_c。假设所有 SID 的集合为 $\{C_1...C_{N_d}\}$，对应的物品计数集合为 $I = \{5, 1, ..., 4\}$ 将 SID 按照其关联的物品数量 I_c 从小到大进行排序，得到排序后的列表 $S_{id} = \{SID_1, ..., SID_{N_d}\}$，其中 N_d 是 SID 的总数。

计算前 i 个 SID 的物品累积数量 $C(i)$：

$$C(i) = \sum_{0}^{i} I_c(S_{id}^i)$$

计算洛伦兹曲线值 $L(i)$：

$$L(i) = \frac{C(i)}{C(N_d)}$$

最终基尼系数 $G = \frac{2}{N_d}\sum_{i=1}^{N_d}(\frac{i}{N_d} - L(i))$

基尼系数越低，表示 SID 的使用越均匀（更公平），通常会导致更好的检索性能，它本质上计算的是均匀分布与实际 SID 分布累积分布函数之间的差异。

3.5 Embedding HitRate [3]

目的是对用于生成 SID 的多模态特征 $\mathcal{H}^i$ 进行初步质量评估，它反映了这些特征是否有效地捕捉了物品间的协同过滤关系。实验表明，嵌入命中率与最终的检索模型性能（HitRate）呈现正相关关系 15。较高的嵌入命中率意味着输入特征包含了高质量的协同信息。

计算过程：将提取出的多模态特征 $\mathcal{H}^i$ 直接视为传统推荐中的 ID Embedding，基于这些多模态特征 $\mathcal{H}^i$ 的相似度，在历史数据上执行 Item-to-Item 的检索。

计算逻辑与标准的 HitRate@K 相同，唯一的区别在于检索结果集 I_K 是仅通过特征 $\mathcal{H}^i$ 的相似度获取的，而非通过模型预测。

$$HR@K = \frac{1}{N}\sum_{m=1}^{N}\frac{I_K \cap I_{click}}{I_{click}}$$

N：测试样本总数。I_{click}：用户实际点击的物品集合。I_K：通过特征相似度检索出的 Top-K 物品集合。

3.6 面向最终的推荐任务评估

这更符合推荐的最终目标，但评估 codebook 的成本更高。本质上 semantic ID 是有损的 embedding，使用 Sid 的目标是希望通过少量的信息损失来换取检索效率的提升。可以对比一下 embedding 的 ANN 对比 Sid 的 beam search 分别检索 topk，分析两种效果的检索效率和检索结果。检索结果可以包括 Hitrate，Recall，NDCG，MRR 这些召回侧的指标，或者更间接更业务的指标，如交易量、video views 等。

3.7 A Practitioner's Handbook

GRID [5] 在 Tokenizer 方面的实验主要揭示了：RK-Means 比复杂的 RQ-VAE 更好；单纯增大预训练模型收益有限；且 SID 的层数不宜过多，需要平衡语义丰富度和序列的可学习性。

更简单的 RK-Means (Residual K-Means) 甚至 R-VQ (Residual Vector Quantization) 在推荐性能（Recall 和 NDCG）上通常优于 RQ-VAE 。RQ-VAE 需要同时训练自编码器和量化器，实施复杂度高且训练困难。实验中即使 RQ-VAE 的训练步数是其他方法的 5 倍，效果依然不如简单的 RK-Means。作者认为 RQ-VAE 带来的性能收益可能不足以抵消其实现的复杂性。

研究测试了使用不同参数规模的 Flan-T5 模型（从 Large 780M 到 XXL 11B）作为语义编码器。尽管 LLM 的参数量增加了 14 倍以上，但下游的推荐性能提升非常微小 (marginal)。这表明当前的“基于 SID 的生成式推荐”范式尚未能充分利用大语言模型中蕴含的丰富世界知识。

实验调节了残差量化的层数 (L) 和每层的 Token 数量/码本大小 (W)。最佳配置：实验发现默认配置 ($L = 3, W = 256$) 效果最好。虽然更多的层数能携带更多的语义信息，但同时也增加了 SID 序列的长度。这揭示了一个权衡关系：SID 序列的可学习性 (Learnability) 与其包含的语义信息量 (Amount of semantic information) 之间存在矛盾。序列过长会增加生成模型的学习难度，从而损害推荐效果。

4 讨论

4.1 Q1:Straight-Through Estimator (STE)

RQ-VAE 的重建损失在反向传播的时候面临的一个障碍是量化操作的不可导性。由于量化过程（即在码本中寻找最近邻 $\arg\min$）是一个离散的阶跃函数，其梯度在几乎所有位置均为 0，这会导致反向传播时梯度断裂，Encoder 无法获得更新信号。为了解决这一问题，RQ-VAE 采用了 Straight-Through Estimator (STE) 。

- **核心思想**：在前向传播中进行真实的离散量化计算，而在反向传播中将量化操作视为恒等映射（Identity Function），使梯度能够“直通”回 Encoder。
- **数学实现**：在计算图中，量化后的输出 $\boldsymbol{z}_{output}$ 通常利用停止梯度操作 $sg[\cdot]$ 构造如下：

$$\boldsymbol{z}_{output} = \boldsymbol{z} + sg[\boldsymbol{z}_q - \boldsymbol{z}] \tag{9}$$

其中：

- $\boldsymbol{z}$ 是 Encoder 输出的连续向量（或某一级的残差）。
- $\boldsymbol{z}_q$ 是量化后的离散码本向量。
- $sg[\cdot]$ 表示停止梯度操作（在前向传播中保持原值，在反向传播中梯度为 0）。

原理解析：

1. **前向传播时**：由于 sg 不影响数值计算，$\boldsymbol{z}_{output} = \boldsymbol{z} + (\boldsymbol{z}_q - \boldsymbol{z}) = \boldsymbol{z}_q$。这保证了后续 Decoder 接收到的是真实的离散量化向量，计算出的重构 Loss 是准确的。
2. **反向传播时**：由于 sg 阻断了括号内的梯度流向，$\frac{\partial(\boldsymbol{z}_q - \boldsymbol{z})}{\partial \boldsymbol{z}}$ 部分被忽略，使得 $\frac{\partial \boldsymbol{z}_{output}}{\partial \boldsymbol{z}} = 1$。这意味着 Loss 对 $\boldsymbol{z}_q$ 的梯度被直接“复制”并传回给 $\boldsymbol{z}$，从而实现了对 Encoder 参数的有效更新。

4.2 Q2: 关键的问题：Codebook 的初始化

实际操作过程中，绕不开的一个问题就是 codebook 的初始化，codebook 的初始化对最终的任务效果有决定性的影响。首先，为什么随机/正态分布初始化通常不好。因为在 Semantic ID 的编码方法中，Codebook 本质上是特征空间中的一组聚类中心。如果使用随机分布初始化 Codebook，这些向量可能分布在整个高维空间中，而实际的 Item Embedding 数据通常只占据空间中的一个小流形（Manifold）。在训练初期，只有少数几个碰巧离数据比较近的 Codebook 向量会被选中并更新。绝大多数 Codebook 向量因为离数据太远，永远不会被“激活”（选中），因此也永远得不到梯度的更新，成为了“死码”（Dead Codes）。结果就是：本该有 8096 个表达能力的码本，最后可能只有 1000 个向量在工作，严重限制了模型的表达能力。

如果还想用随机初始化，可以使用方差较大的正态分布可以缓解这个利用率较低的现象，确实是一个启发式的算法，这相当于把码本向量“炸开”，散布到更广阔的空间中。这增加了某些码本向量恰好落在数据分布区域（或其附近）的概率。分散的向量将空间切割成更大的 Voronoi 单元。这意味着每个向量“管辖”的空间体积变大了，它捕获到数据点的概率也随之增加。只要它捕获到哪怕一个数据点，它就能通过梯度“复活”并移动到正确的位置。

而更好的方法是像 TIGER 中论文明确提出，不使用随机初始化，而是使用 K-Means 初始化。在训练开始前，先拿第一个 Batch（或前几个 Batch）的 Item Embedding 数据跑一次 K-Means 聚类，将得到的“聚类中心”直接赋值给 Codebook 作为初始权重。这保证了 Codebook 向量一开始就分布在数据密集的区域，每个向量都有机会被选中，从而避免了坍塌，保证了高码本利用率。

而 Onerec [2] 的 K-Means 算法本身就是一个迭代寻找中心的过程。它不像 VQ-VAE 那样是一个固定的参数矩阵等着梯度来更新，而是每一步（或每一轮）都在根据数据重新计算中心。这种非梯度的聚类方法天然规避了“初始化太远导致梯度传不过去”的问题。FORGE 则是引入了 Gini 系数来监控码本的负载均衡，确保没有 Codebook 被闲置。

4.3 Q3: 如何应对新的 item

首先，需要区分一个概念，有新的 item，我们的目标是推理还是训练。如果只是需要推理出新 item 的 semantic ID，那么无论是 res-kmeans 还是 rq-vae 都是可以直接利用 codebook，推理出物品的 semantic ID。如果是要根据新 item 来训练 codebook，那么 res-kmenas 由于需要保持不同簇的数量分布均匀，会更新簇中心（即 codebook），甚至会更新别的 item 的 semantic ID。但是 RQ-VAE 同样也会更新 codebook（由于 codebook loss 的存在），甚至也会更新别的 item 的 semantic ID。

所以在有新物品时，无论是 rq-vae 还是 rq-kmenas，两者在训练和推理时，对 codebook 和别的物品的 semantic ID 的影响都是相近的。在这里讨论这个问题是由于我发现很多人在刚接触 Res-kmeans

都会以为为了这个簇的数量分布均匀，新物品出现会造成算法不稳定的问题。实际上，在部署这个方法的时候，也是会每天进行增量更新训练的，但不是只要出现一个新物品就会重新训练。

4.4 Q4:LSH（Locality Sensitive Hashing）

在 TIGER 提出之前，工业界处理海量 Item ID 的一种常见思路就是哈希（Hashing）。LSH 作为 Baseline 有非常明确的对比意义：代表“非学习”方法：LSH 不需要训练，它的映射规则是随机生成的，与数据分布无关。对比 LSH 可以证明“学习数据的分布（如 RQ-VAE）”是否真的比“随机几何划分”更有效。保序性（Locality Preserving）：SimHash 理论上保证了：如果原向量 **x** 和 **y** 的余弦相似度高，它们的 Hash 值相等的概率就高。这符合 Semantic ID “相似物品有相似 ID”的核心诉求。计算极其廉价：只需要做矩阵乘法，非常适合大规模系统。

SimHash 的核心直觉是如果两个向量在高维空间中非常接近（夹角很小），那么用随机的超平面去切割空间时，这两个向量大概率会落在超平面的同一侧。想象一个西瓜（高维特征空间），我们随机切一刀（一个超平面）：西瓜子 A 和西瓜子 B 如果靠得很近，它们很有可能在刀的同一侧。如果我们要给西瓜子编码，就可以记录它在每一刀的“左边”还是“右边”（0 或 1）。切很多刀之后，得到的一串 01 代码就是它的 Hash 值。

为了生成一个离散的 Token，TIGER 使用了 h 个随机超平面。参数：$w_1, w_2, ..., w_h$ 是随机采样的向量（从标准正态分布采样，以保证球形对称性）。TIGER 设置：$h = 8$。这意味着每个 Token 本质上是一个 8-bit 的整数（取值范围 0-255）。

对于物品的语义嵌入向量 **x**，计算它与每个超平面的内积，并根据符号决定是 0 还是 1：如果 **x** 在超平面 w_i 的正侧，记为 1。如果在负侧，记为 0。这样就得到了一个长度为 h 的二进制向量：$(b_1, b_2, ..., b_h)$。

最后将上述二进制向量转换为一个十进制整数 c，作为一个 Token：

$$c = \sum_{i=1}^{h} 2^{i-1} \cdot b_i$$

例如，如果 $h = 8$，结果就是一个 0 255 之间的整数。由此，每一个物品的 ID 就用一个整数代替。

然后这些由整数组成的 seq 输入给 lookup table 转换成 embedding，再把这个 seq 输入给 transormer，端到端的学习 lookup table 和 transformer。

4.5 Q5:Res-Kmeans 分布式场景

簇 k 的分配结果直接改变了簇 $k+1$ 的候选集 $\mathcal{U}$。这意味着该算法无法同时处理 Cluster 1 和 Cluster 2。如果 $K = 8192$（常见配置），就需要串行循环 8192 次，且每次都要对海量数据进行排序，这在分布式环境（如 Spark/MapReduce）中会导致极高的通信和调度开销。

算法小小怪下士不是大数据工程的专家，但思考了一些可能的解决方案：

1. 首先，可能单机就能 load 住。虽然感觉数据量很大，但想想可能还好。如果是 1 亿个物品，维度是 1024，float32 向量，大约是 512GB 可以 handle 住的，放入单机内存应该问题也不大。OneRec 的技术报告提到训练使用了 90 台服务器，但 Tokenizer 的生成（离线过程）通常不需要这么复杂的集群，而且使用了 RQ-Kmeans 相比 RQ-VAE 减少了计算量。

2，如果真的大到单机无法承受，那么就像 GRID 中写的，也可以用 Residual Mini-Batch k—means，Mini-Batch 更支持数据并行。

4.6 Q6:VAE 的来时路

AE 的目标是将输入 $\boldsymbol{x}$ 压缩为低维向量 $\boldsymbol{z}$，再将其还原为 $\hat{\boldsymbol{x}}$。其潜在空间（Latent Space）是确定性的点对点映射。

AE 的损失函数非常直观，仅包含 Reconstruction Loss)。它衡量解码输出与原始输入之间的差异。

$$\mathcal{L}_{AE} = \underbrace{\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2}_{\text{MSE Loss}} \tag{10}$$

或者在输入为二值图像时使用交叉熵：

$$\mathcal{L}_{AE} = -\sum_{i} [x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i)] \tag{11}$$

- **目标**：仅仅追求“画得像”。
- **潜在空间**：由于缺乏正则化项，AE 的潜在空间是不连续的。模型可能会死记硬背训练数据的位置，导致潜在空间中存在大量“空白区域”，不适合用于生成新样本。

VAE 认为潜在变量 $\boldsymbol{z}$ 服从某种分布。Encoder 不再直接输出 $\boldsymbol{z}$，而是输出分布的参数（均值 $\boldsymbol{\mu}$ 和方差 $\boldsymbol{\sigma}^2$），然后通过重参数化（Reparameterization Trick）采样得到 $\boldsymbol{z}$。

VAE 的损失函数推导来自 Evidence Lower Bound, ELBO。它由两部分组成：重构项和正则化项。

$$\mathcal{L}_{VAE} = \underbrace{\mathcal{L}_{Recon}}_{\text{重构质量}} + \beta \cdot \underbrace{\mathcal{L}_{KL}}_{\text{分布正则化}} \tag{12}$$

在实际工程实现中，重构损失与 AE 的第一部分相同。

$$\mathcal{L}_{Recon} = -\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] \tag{13}$$

KL 散度，它约束 Encoder 输出的后验分布 $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ 尽可能接近先验分布 $p(\boldsymbol{z})$。

$$\mathcal{L}_{KL} = D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})) \tag{14}$$

对于高斯分布，KL 散度有解析解：

$$\mathcal{L}_{KL} = -\frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2\right) \tag{15}$$

VAE 的训练过程本质上是两个 Loss 的博弈：重构损失希望 $\boldsymbol{\sigma} \to 0$。因为方差越小，采样越确定，重构越容易（退化回 AE）。KL 损失希望 $\boldsymbol{\mu} \to 0, \boldsymbol{\sigma} \to 1$。它强迫潜在空间变得平滑、连续并服从正态分布。正是这种博弈，使得 VAE 的潜在空间既保留了数据特征，又具备了良好的生成性质（连续性）。

特性	**AE**	**VAE**
映射方式	$\boldsymbol{x} \to \boldsymbol{z}$ (点到点)	$\boldsymbol{x} \to \mathcal{N}(\mu, \sigma) \to \boldsymbol{z}$ (点到分布)
损失函数	重构误差	重构误差 + **KL 散度**
潜在空间	不连续，易过拟合	连续，平滑，适合生成

4.7 Q7: 沙漏现象

沙漏现象是指 codeboook token 过度集中，形成了一对多和多对一的映射。这种集中导致了路径稀疏性，item 的匹配路径仅占整个路径的极小部分。

为什么会出现沙漏现象？由于第一层的残差值的大小差异，该层的输入分布是非均匀的。存在大量较小幅度的点（来自前一层的每个桶中靠近簇中心的点），同时，存在少量大幅度的点，这些点被视为异常值。为了减少聚类损失，该层的聚类过程重点关注这些异常值。因此，较小幅度的点将占据较少的簇中心，而异常值将要么占据单个簇中心，要么占据多个簇中心。因此，该层的语义 ID 将形成较大的路由结点，呈现出长尾现象。

为缓解沙漏效应，京东的论文提出了两种方法，第一种是直接移除第二层，第二种是通过实施自适应 token 分布调整，从第二层剔除顶部 token，从而将予以 ID 替换为变长结构。

5 Appendix

5.1 手撕 RQ-VAE

5.2 手撕 RQ-kmeans

5.3 手撕 FSQ

参考文献

[1] Chen, B., Guo, X., Wang, S., Liang, Z., Lv, Y., Ma, Y., Xiao, X., Xue, B., Zhang, X., Yang, Y., Dai, H., Xu, X., Zhao, T., Peng, M., Zheng, X., Wang, C., Zhao, Q., Zhai, Z., Zhao, Y., Liu, B., Lv, J., Liang, X., Ding, Y., Chen, J., Lei, C., Ou, W., Li, H., and Gai, K. Onesearch: A preliminary exploration of the unified end-to-end generative framework for e-commerce search, 2025.

[2] Deng, J., Wang, S., Cai, K., Ren, L., Hu, Q., Ding, W., Luo, Q., and Zhou, G. Onerec: Unifying retrieve and rank with generative recommender and iterative preference alignment, 2025.

[3] Fu, K., Zhang, T., Xiao, S., Wang, Z., Zhang, X., Zhang, C., Yan, Y., Zheng, J., Li, Y., Chen, Z., Wu, J., Kong, X., Zhang, S., Kuang, K., Jiang, Y., and Zheng, B. Forge: Forming semantic identifiers for generative retrieval in industrial datasets, 2025.

[4] Jiang, Y., Ren, X., Xia, L., Luo, D., Lin, K., and Huang, C. Recgpt: A foundation model for sequential recommendation, 2025.

[5] Ju, C. M., Collins, L., Neves, L., Kumar, B., Wang, L. Y., Zhao, T., and Shah, N. Generative recommendation with semantic ids: A practitioner's handbook, 2025.

[6] Kuai, Z., Chen, Z., Wang, H., Li, M., Miao, D., Wang, B., Chen, X., Kuang, L., Han, Y., Wang, J., Tang, G., Liu, L., Wang, S., and Zhuo, J. Breaking the hourglass phenomenon of residual quantization: Enhancing the upper bound of generative retrieval, 2024.

[7] Rajput, S., Mehta, N., Singh, A., Keshavan, R. H., Vu, T., Heldt, L., Hong, L., Tay, Y., Tran, V. Q., Samost, J., Kula, M., Chi, E. H., and Sathiamoorthy, M. Recommender systems with generative retrieval, 2023.

[8] Wang, W., Bao, H., Lin, X., Zhang, J., Li, Y., Feng, F., Ng, S.-K., and Chua, T.-S. Learnable item tokenization for generative recommendation, 2025.

[9] Wang, Y., Gan, W., Xiao, L., Zhu, J., Chang, H., Wang, H., Zhang, R., Dong, Z., Tang, R., and Li, R. Act-with-think: Chunk auto-regressive modeling for generative recommendation, 2025.

[10] Xu, Y., Fan, C., Hu, J., Zhang, Y., Xiaoyi, Z., and Zhang, J. Store: Semantic tokenization, orthogonal rotation and efficient attention for scaling up ranking models, 2025.

[11] Yao, Y., Li, Z., Xiao, S., Du, B., Zhu, J., Zheng, J., Kong, X., and Jiang, Y. Saviorrec: Semantic-behavior alignment for cold-start recommendation, 2025.

[12] Zhou, G., Deng, J., Zhang, J., Cai, K., Ren, L., Luo, Q., Wang, Q., Hu, Q., Huang, R., Wang, S., Ding, W., Li, W., Luo, X., Wang, X., Cheng, Z., Zhang, Z., Zhang, B., Wang, B., Ma, C., Song, C., Wang, C., Wang, D., Meng, D., Yang, F., Zhang, F., Jiang, F., Zhang, F., Wang, G., Zhang, G., Li, H., Hu, H., Lin, H., Cheng, H., Cao, H., Wang, H., Huang, J., Chen, J., Liu, J., Jia, J., Gai, K., Hu, L., Zeng, L., Yu, L., Wang, Q., Zhou, Q., Wang, S., He, S., Yang, S., Yang, S., Huang, S., Wu, T., He, T., Gao, T., Yuan, W., Liang, X., Xu, X., Liu, X., Wang, Y., Wang, Y., Liu, Y., Song, Y., Zhang, Y., Wu, Y., Zhao, Y., and Liu, Z. Onerec technical report, 2025.