Multimedia Systems (2023) 29:1499–1512
https://doi.org/10.1007/s00530-023-01065-2

REGULAR PAPER

Resizing codebook of vector quantization without retraining

Lei Li[1] · Tingting Liu[1] · Chengyu Wang[3] · Minghui Qiu[3] · Cen Chen[1,2] · Ming Gao[1,2] · Aoying Zhou[1]

Received: 20 June 2022 / Accepted: 10 February 2023 / Published online: 7 March 2023

Abstract

Large models pre-trained on massive data have become a flourishing paradigm of artificial intelligence systems. Recent works, such as M6, CogView, WenLan 2.0, NÜWA, and ERNIE-ViLG, further extend this diagram to joint Vision Language Pre-training (VLP). For VLP, the two-stage architecture is a popular design, which includes the first stage learning an encoding function of data and the second stage learning a probabilistic model of encoded representation of data. Vector quantization (VQ) has usually engaged in the encoding function of image data for the first stage. VQ includes a data structure (codebook) and an algorithm (finding nearest quantization). The publicly available VQ models (e.g., VQGAN, VQVAE, VQVAE2) include a codebook whose size is assigned empirically (e.g., 1024, 4096, and 16,384) by their authors. If we want a smaller codebook for a lower computation load of the VQ process, or we want a larger codebook for better reconstruction quality, we have to retrain VQ models that consist of the down-sampling net, the codebook, and the up-sampling net. However, retraining VQ models is very expensive since these models, with billions of parameters, are trained on massive datasets. It motivates us to find an approach to resize the codebook of Vector quantization without retraining. In this paper, we leverage hyperbolic embeddings to enhance codebook vectors with the co-occurrence information and reorder the enhanced codebook by the Hilbert curve. Then we can resize the codebook of vector quantization for lower computation load or better reconstruction quality. Experimental results prove the efficiency and effectiveness of our approach when compared with competitive baselines. The code will be released to the public.

Keywords Codebook resizing · Vector quantization · Hyperbolic embeddings · Hilbert curve

1 Introduction

Large models pre-trained on massive data have become a flourishing paradigm of artificial intelligence systems. BERT [5] and GPT [27] grow in popularity in the natural language processing community as they possess high transferability to a wide range of downstream tasks, yielding state-of-the-art performance. Recent works, such as M6 [20], CogView [7], WenLan 2.0 [9], NÜWA [35], and ERNIE-ViLG [39], further extend this diagram to the joint Vision Language Pre-training (VLP) domain and show superior results over state-of-the-art methods on various downstream tasks. For VLP, the two-stage architecture is a popular design, which includes the first stage learning an encoding function of data and the second stage learning a probabilistic model of encoded representation of data.

Vector Quantization (VQ) has usually engaged in the encoding function of image data for the first stage. VQ includes a data structure (codebook) and an algorithm (finding nearest quantization). Equipping VQ with

✉ Cen Chen
cenchen@dase.ecnu.edu.cn

Lei Li
leili@stu.ecnu.edu.cn

Tingting Liu
ttliu@stu.ecnu.edu.cn

Chengyu Wang
chengyu.wcy@alibaba-inc.com

Minghui Qiu
minghuiqiu@gmail.com

Ming Gao
mgao@dase.ecnu.edu.cn

Aoying Zhou
ayzhou@dase.ecnu.edu.cn

1 Shanghai Engineering Research Center of Big Data Management, School of Data Science and Engineering, East China Normal University, Shanghai 200062, China

2 KLATASDS-MOE, School of Statistics, East China Normal University, Shanghai 200062, China

3 Alibaba Group, Hangzhou 311121, Zhejiang, China

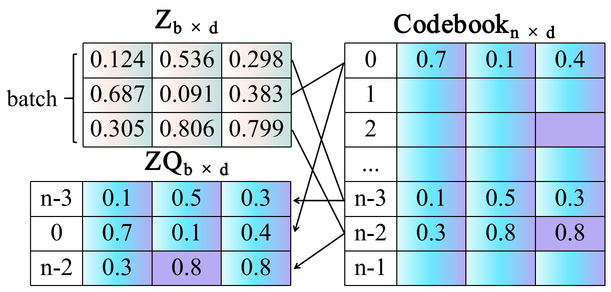

Fig. 1 A visualization of VQ. During VQ, a batch of latent vectors find the nearest quantization vectors and their corresponding indices from the codebook

different loss functions and down/up-sampling techniques, we can obtain various encoding functions, such as VQGAN [8], VQVAE [25], and VQVAE2 [28]. Figure 1 presents a visualization of VQ where $\boldsymbol{Z}_{b\times d}$ is a batch of vector, b is the batch size, and d is the dimensionality of a vector. Furthermore, we have a codebook $\boldsymbol{Codebook}_{n\times d}$, which contains n vectors with the dimensionality as d. For each vector in $\boldsymbol{Z}_{b\times d}$, VQ aims at finding the nearest vector (Euclidean distance) from $\boldsymbol{Codebook}_{n\times d}$. $\boldsymbol{ZQ}_{b\times d}$ consisting of these quantized replacements will engage in subsequent computation involving $\boldsymbol{Z}_{b\times d}$. Indices i.e., $(n-3, 0, n-2)$ serve as discrete tokens for vectors in $\boldsymbol{Z}_{b\times d}$.

The publicly a-vailable VQ models (e.g., VQGAN, VQVAE, VQVAE2) include a codebook whose size is assigned empirically (e.g., 1024, 4096, and 16,384) by their authors. If we want a smaller codebook for a lower computation load of the VQ process, or we want a larger codebook for better reconstruction quality, we have to retrain VQ models that consist of the down-sampling net, the codebook, and the up-sampling net. However, retraining VQ models is very expensive since these models, with billions of parameters, are trained on massive datasets, i.e., ImageNet with 14,197,122 images. It motivates us to find an approach to resize the codebook of Vector Quantization without retraining. To achieve this goal, we make the following contributions:

- We leverage hyperbolic embedding to enhance codebook vectors with the co-occurrence information and logical similarities since hyperbolic embedding is proved more effective than euclidean embedding for learning latent semantics of images [18]. Hyperbolic embedding enhanced codebook entries will get closer in the high-dimensional space than the original ones.
- For users who need a smaller codebook size to reduce computation load, we employ the Hilbert curve to reorder the hyperbolic embedding enhanced codebook, preserving the locality of vectors. Then we can use a smaller subset of the codebook to achieve similar reconstruction metrics compared to the original VQ.
- For users who need a larger codebook size for better reconstruction quality, we can conduct interpolation between each pair of adjacent entries in the reordered codebook.
- Extensive experiments show the effectiveness of our approach compared to various baselines.

The rest of this paper is summarized as follows: Sect. 2 introduces related works, including vector quantization, hyperbolic embedding, and Hilbert curve. Section 3 present our methodology, which contains three significant steps, e.g., hyperbolic embedding enhancements, reordering by Hilbert curve, and resizing the codebook without retraining. Section 4 consists of extensive experimental results which prove the effectiveness and efficiency of our proposed approaches. Section 5 draws comprehensive conclusions from this paper, and offers several interesting ideas for future work.

2 Related work

2.1 Vector quantization

Vector quantization (VQ) is widely used as the discrete encoding approach. To avoid the “neighbor explosion” problem of GNNs, Ding et al. [6] propose a universal framework to scale up any convolution-based GNNs using Vector Quantization without compromising the performance. Roy and Grangier [29] propose a residual variant of vector-quantized variational auto-encoder to learn paraphrasing models from an unlabeled monolingual corpus only. van den Oord et al. [25] propose VQVAE to learn a discrete latent representation. Using the VQ method allows the model to circumvent issues of “posterior collapse” (where the latent information is ignored when they are paired with a powerful autoregressive decoder) typically observed in the VAE framework. Razavi et al. [28] demonstrate that VQVAE2 (a multi-scale hierarchical organization of VQ-VAE), augmented with powerful priors over the latent codes, is able to generate samples with quality that rivals that of state-of-the-art Generative Adversarial Networks (GAN) on multifaceted datasets. VQGAN [8] utilizes VQ to improve the GAN training and yields improved performance. VQGAN also encoding images into discrete tokens for multi-modal models e.g., M6 [20], CogView [7], WenLan 2.0 [9], NÜWA [35], and ERNIE-ViLG [39].

Vector Quantization (VQ) is also integrated with many non-deep-learning approaches. (1) Quantization-based techniques are the current state-of-the-art for scaling maximum inner product search to massive databases. Based on the observation that for a given query, the database points that have the largest inner products are more relevant, Guo et al. [12] develop a family of anisotropic quantization loss functions. These functions lead to a new variant of vector quantization that more greatly penalizes the parallel component of a datapoint's residual relative to its orthogonal component. (2) Ai et al. [2] propose an optimized residual vector quantization-based approach for improving the quality of vector quantization and approximate nearest neighbor search. Based on residual vector quantization (RVQ), a joint optimization process called enhanced RVQ (ERVQ) is introduced. Each stage codebook is iteratively optimized by the others aiming at minimizing the overall quantization errors. To efficiently find the nearest centroids when quantizing vectors, a non-linear vector quantization method is proposed. The vectors are embedded into two-dimensional space where the lower bounds of Euclidean distances between the vectors and centroids are calculated. The lower bound is used to filter non-nearest centroids for the purpose of reducing computational costs. (3) [36] propose a multi-scale quantization approach for fast similarity search on large, high-dimensional datasets. The key insight of their approach is that quantization methods, in particular product quantization, perform poorly when there is a large variance in the norms of the data points. This is a common scenario for real-world datasets, especially when doing product quantization of residuals obtained from coarse vector quantization. To address this issue, they propose a multiscale formulation that learns a separate scalar quantizer of the residual norm scales.

2.2 Hyperbolic embedding

There has been an emerging trend for deep learning in hyperbolic spaces since they possess high capacity and ability of modeling hierarchical structure. Hyperbolic space is a homogeneous space with constant negative curvature. It is a smooth Riemannian manifold and, as such, locally Euclidean space. The hyperbolic space can be modeled using five isometric models [26], which are the Lorentz (hyperboloid) model, the Poincaré ball model, Poincaré half-space model, the Klein model, and the hemisphere model.

As far as we know, the work [24] is the first to propose learning an embedding using Poincaré model while considering the latent hierarchical structures. They also proved that Poincaré embeddings could outperform Euclidean embeddings significantly on data with latent hierarchies, both in terms of representation capacity and in terms of generalization ability. Khrulkov et al. [18] have shown that across a number of tasks, in particular in the few-shot image classification, learning hyperbolic embeddings can result in a substantial boost in accuracy. They speculate that the negative curvature of the hyperbolic spaces allows for embeddings that are better conforming to the intrinsic geometry of at least some image manifolds with their hierarchical structure. Yang et al. [38] bring up a Hyperbolic Regularization powered Collaborative Filtering (HRCF) and design a geometric-aware hyperbolic regularizer. Specifically, the proposal boosts the optimization procedure via the root alignment and origin-aware penalty, which is simple yet impressively effective. Bai et al. [3] present ConE (Cone Embedding), a KG embedding model that is able to simultaneously model multiple hierarchical as well as non-hierarchical relations in a knowledge graph. ConE embeds entities into hyperbolic cones and models relations as transformations between the cones. In particular, ConE uses cone containment constraints in different subspaces of the hyperbolic embedding space to capture multiple heterogeneous hierarchies. Wang et al. [33] propose a fully hyperbolic GCN model for the recommendation, where all operations are performed in hyperbolic space. Utilizing the advantage of hyperbolic space, their method is able to embed users/items with less distortion and capture user–item interaction relationships more accurately. Extensive experiments on public benchmark datasets show that their method outperforms both Euclidean and hyperbolic counterparts and requires far lower embedding dimensionality to achieve comparable performance.

2.3 Hilbert curve

In mathematical analysis, a space-filling curve is a curve whose range contains the entire two-dimensional unit square, more generally a unit hypercube in n-dimensional space. The "Hilbert curve" is a specific curve which covers the interior of the n-dimensional hypercube $[0, 2^p)^n$ of side 2^p with unit precision. Explicitly using an integer grid absolves us from formal technicalities of the continuum limit: we cannot store infinite precision in our physical hardware.

Chen and Chang [4] propose the one-nearest-neighbor finding strategy directly based on the Hilbert curve. By relations among orientations, orders, and quaternary numbers, they compute the relative locations of the query block and the neighboring block in the Hilbert curve. Then, the nearest neighbor of one query point can be found directly from these neighboring blocks.

Tsinganos et al. [32] utilize the Hilbert space-filling curve for the generation of image representations of surface electromyography (sEMG) signals that are then classified by CNN. The proposed method is evaluated on different network architectures and yields a classification improvement of more than 3%.

Wu et al. [37] propose a spatiotemporal index method based on the Hilbert curve code. It is more efficient than the existing spatiotemporal index method and can effectively support the management of massive multi-scale trajectory data.

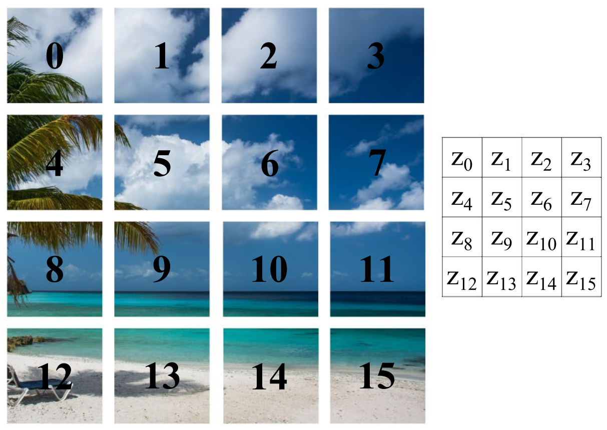

Fig. 2 After four downsampling steps, an input image of size 64×64 will be mapped to a latent embeddings of size 4×4

3 Methodology

We name our approach **HyperHill** which includes three phases:

1. enhancing codebook entries with co-occurrence information provided by hyperbolic embedding;
2. reordering the codebook by the Hibert curve;
3. resizing the codebook without retraining.

3.1 Hyperbolic embeddings enhanced codebook

As shown in Fig. 2, we perform four downsampling steps. For example, an input image of size 64×64 will be mapped to 4×4 latent embeddings z_0 to z_{15}. As aforementioned, these 16 embeddings serve as a batch in Fig. 1 that engages in a matrix multiplication with the whole codebook. To reduce the computation load of VQ, we hope that we can only use a subset of the codebook to finish vector quantization. Thus, we need to cluster similar entries in the codebook and develop a method to query the subset of the codebook.

Based on Fig. 2, we observe two kinds of similarities. We denote the first similarity as the **perceptual similarity** which indicates image blocks are similar, i.e., block 10 (sky) and block 11 (sky). We denote the second similarity as the **logical similarity** which indicates image blocks are more probable to co-occur adjacently in a small patch of the image, i.e., block 10 (sky) and block 6 (cloud). **Perceptual similarity** is already modeled by the latent embeddings after downsampling. However, **logical similarity** has not been considered by previous VQ works. Besides **perceptual similarity**, it motivates us to build another embedding for image tokens based on the **logical similarity**. Such **logical similarity** embedding should make image tokens that are more probable to co-occur adjacently in a small patch of the image to get closer in the embedding space. We employ the Hyperbolic embedding for building **logical similarity** embedding of image tokens for two reasons:

1. In many image-related tasks, such as image classification, image retrieval, and few-shot learning, Khrulkov et al. [18] demonstrates that hyperbolic embeddings provide better representations than linear hyperplanes, Euclidean distances, or spherical geodesic distances.
2. Hyperbolic embedding is also able to build embedding with fewer dimensions than euclidean embedding, keeping similar downstream task performance [26]. Fewer dimensions of image token embedding will reduce the computation complexity of reordering entries of the VQ codebook. It should be noted that we do not guarantee the existence of or rely on hierarchical structures of image tokens.

As an exploratory work, our contribution is defining the training task as pulling close co-occurred adjacently image tokens and pushing away non-co-occurred adjacently image tokens in the Hyperbolic space. Experimental results prove that our training task is capable of modeling **logical similarity**, and **logical similarity** can be integrated with **perceptual similarity** by the vector concatenation, to get better representations of image tokens for the reordering. In future work, we plan to inspect more embedding approaches for modeling **logical similarity**, further reduce dimensions of **logical similarity** embedding, and introduce new techniques (e.g., cross-attention and multi-task learning) for the fusion of **perceptual similarity** and **logical similarity**.

Hyperbolic geometry [26] is a non-Euclidean geometry that studies spaces of constant negative curvature. We choose the Poincaré ball model [24] among hyperbolic models since it is well-suited for gradient-based optimization. Figure 3 is a visualization showing the two-dimensional Poincaré ball model of hyperbolic geometry. The entire geometry is located within the unit circle. Hyperbolic lines are actually arcs of a circle that intersect at right angles to the unit circle. Two hyperbolic lines in Fig. 3 have the same hyperbolic length since the same Euclidean length near the edge of the circle is much longer than near the center.

This is due to the hyperbolic geometry, which has a very different distance function. Let

$$\mathcal{B}^d = \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid \|\boldsymbol{x}\| < 1 \right\}, \tag{1}$$

be the *open* d-dimensional unit ball, where $\|\cdot\|$ denotes the Euclidean norm. The Poincaré ball model of hyperbolic space corresponds to the Riemannian manifold $(\mathcal{B}^d, g_x)$, i.e., the open unit ball equipped with the Riemannian metric tensor

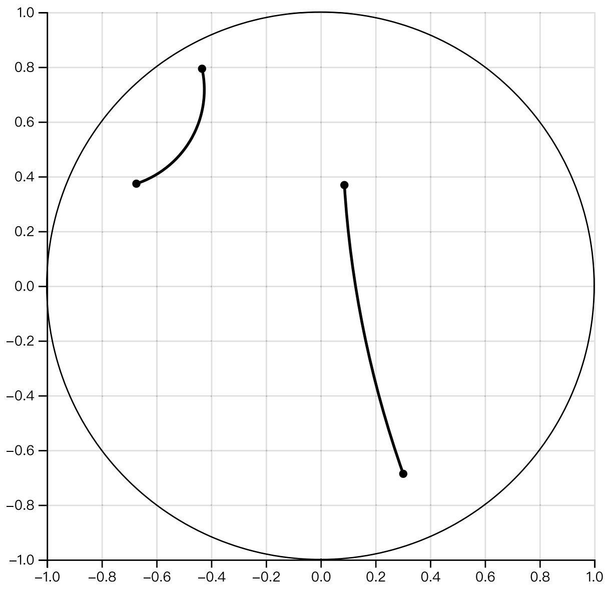

Fig. 3 A visualization shows the two-dimensional Poincaré ball model of hyperbolic geometry. Two hyperbolic lines have the same hyperbolic length since the same Euclidean length near the edge of the circle is much longer than near the center

$$g_x = \left(\frac{2}{1-\|x\|^2}\right)^2 g^E, \tag{2}$$

where $x \in \mathcal{B}^d$ and g^E denote the Euclidean metric tensors. The distance between points $u, v \in \mathcal{B}^d$ is given as

$$d(u, v) = \operatorname{arcosh}\left(1 + 2\frac{\|u - v\|^2}{(1-\|u\|^2)(1-\|v\|^2)}\right). \tag{3}$$

The boundary of the ball is denoted by $\partial\mathcal{B}$. It corresponds to the sphere $\mathcal{S}^{d-1}$ and is not part of the hyperbolic space but represents infinitely distant points. Refer to Peng et al. [26] for more details about hyperbolic geometry.

Poincaré ball model has been proved to be effective on organizing objects in large datasets according to a latent hierarchy. Nickel and Kiela [24] succeed to leverage Poincaré ball model to model natural language tokens (i.e., WordNet). Khrulkov et al. [18] employ Poincaré ball model to finish image tasks i.e., few-shot classification and person re-identification. To compute Poincaré embeddings for a set of VQ indices

$$\mathcal{S} = \{x_i\}_{i=1}^n, \tag{4}$$

we need to find embeddings

$$\Theta = \{\theta_i\}_{i=1}^n, \text{where } \theta_i \in \mathcal{B}^d \tag{5}$$

To estimate Θ, we solve the optimization problem:

$$\Theta' \leftarrow \arg\min_{\Theta} \mathcal{L}(\Theta) \qquad \text{s.t. } \forall\, \theta_i \in \Theta : \|\theta_i\| < 1, \tag{6}$$

where $\mathcal{L}(\Theta)$ represents the loss function which encourages VQ indices that co-occur probably when we encode an image, to be close in the embedding space according to their Poincaré distance. Details of $\mathcal{L}(\Theta)$ is presented in Formula (7)

$$\mathcal{L}(\Theta) = \sum_{(u,v)\in \mathbf{D}} -\log \frac{e^{-d(u,v)}}{\sum_{v' \in \mathcal{N}(u)} e^{-d(u,v')}}, \tag{7}$$

where $\mathbf{D} = \{(u, v)\}$ is the set of observed co-occurrences between VQ indices.

$$\mathcal{N}(u) = \{v \mid (u, v) \notin \mathbf{D}\} \cup \{u\}, \tag{8}$$

is the set of negative examples for u (including u).

Algorithm 1 Poincaré Embedding of VQ Indices

```
1: $\mathbf{D} \leftarrow \emptyset$
2: for image in $Dataset_{train}$:
3:     indices = VQ(encode(image))
4:     for $\boldsymbol{u}$ in indices:
5:         for $\boldsymbol{v}$ in neighbours($\boldsymbol{u}$):
6:             $\mathbf{D} \leftarrow \mathbf{D} \cup \{(\boldsymbol{u},\boldsymbol{v})\}$
7: for one_epoch in epochs:
8:     for one_batch in batches:
9:         $\boldsymbol{\theta}_{t+1} = proj\left(-\eta_t \times g_x^{-1}(\nabla_E)\right)$
```

We employ Algorithm 1 to show the full process of Poincaré embedding of VQ indices. From Line 1 to 6, we traverse each image in the training dataset. The encode function conducts downsampling steps, and the input image is mapped to latent embeddings. Then we leverage VQ to get indices of latent embeddings.

For each index, we find its adjacent neighbors e.g., eight neighbors of z_5 in Fig. 2 are $z_0, z_1, z_2, z_4, z_6, z_8, z_9, z_{10}$ and three neighbors of z_0 in Fig. 2 are z_1, z_4, z_5. Then we add these observed co-occurrences between VQ indices into **D**. Line 7 to Line 9 indicate the training process of Poincaré embedding [24]. η_t denotes the learning rate at time t and the projection function constrains the embeddings to remain within the Poincaré ball

$$\operatorname{proj}(\theta) = \begin{cases} \theta/\|\theta\| - \varepsilon & \text{if } \|\theta\| \geq 1 \\ \theta & \text{otherwise}, \end{cases} \tag{9}$$

where ε is a small constant (i.e., 10^{-5}) to ensure numerical stability. ∇_E is the Euclidean gradient of Formula (7). As aforementioned, g_x^{-1} is inverse of Formula (2) and rescale ∇_E since both distance and gradient are extremely large near the edge of the Poincaré ball.

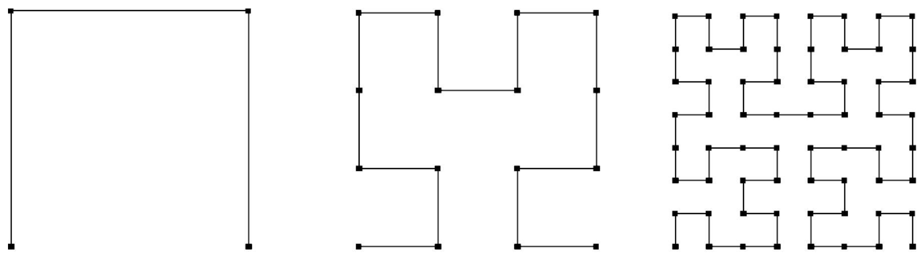
Fig. 4 Hilbert curves in two dimensions for $p = 1, 2, 3$

Assuming that entries (i.e., embeddings) in the codebook of VQ are d-dimensional, the corresponding Poincaré embedding of each entry is k-dimensional. Then we construct a hybrid embedding

$$e^{d+k} = \text{concatenate}\left(z_q{}^d, \boldsymbol{\theta}^k\right) \tag{10}$$

that captures information of both perceptual similarity and logical similarity.

Furthermore, we use VQGAN to encode images in each dataset used in our experiments, and we observe that $z_q{}^d$ (produced by VQGAN) almost always lies within the range $(-1, 1)$ so that we do not introduce extra normalization before concatenating $z_q{}^d$ and $\boldsymbol{\theta}^k$. Some entries of the codebook are not used by the training dataset and do not own Poincaré embedding $\boldsymbol{\theta}^k$. For these unseen entries, we let corresponding $\boldsymbol{\theta}^k$ be a k-dimensional uniform distribution on the interval $(-1, 1)$.

3.2 Reordering codebook with Hilbert curve

The hyperbolic embedding enhanced codebook of VQ is still unordered, so we have to search the whole codebook to find the nearest quantization for a given latent vector. Suppose we can build a reordered codebook that preserves the locality of vectors, then we can reduce the search space for quantization.

To achieve this, there are many previous competitors [10, 11, 13, 16, 23] focusing on the approximate nearest neighbor search. However, we consider utilizing the Hilbert curve for the following reasons.

1. The Hilbert curve does not need training, unlike clustering approaches, i.e., k-means.
2. The results produced by the Hilbert curve are consistent, unlike random projection-based locality-sensitive hashing.
3. The Hilbert curve does not introduce extra data structures (e.g., trees and lookup tables) since we need to let the codebook be a contiguous tensor with the automatic gradient.

The "Hilbert curve" is a specific curve which covers the interior of the n-dimensional hypercube $[0, 2^p)^n$ of side 2^p with unit precision. Distance H along the Hilbert curve is representable by a np-bit integer, decomposable as p digits of n bits each. Figure 4 shows Hilbert curves in two dimensions for $p = 1, 2, 3$. Hilbert curve can be constructed recursively by:

1. making four replicates of current curve;
2. rotating the left-bottom replicate 90° clockwise;
3. rotating the right-bottom replicate 90° anticlockwise;
4. adding three lines to connect four replicates.

We follow Skilling [31] to conduct an efficient implementation to convert one dimensional distance H along a Hilbert curve into n-dimensional points, $(x_0, x_1, ...x_{n-1})$, and vice versa. Let

$$G = g_0 g_1 \dots g_{np-1} = H \oplus \lfloor H/2 \rfloor, \tag{11}$$

where g_i represent a bit and g_{np-1} is the lowest-order bit. $\oplus$ is a single exclusive-OR instruction.

Collect these np bits into n preliminary p-bit integers

$$x_i = g_i g_{i+n} g_{i+2n} \cdots g_{i+(p-1)n} \tag{12}$$

for $i = 0, 1, \dots, n-1$. For example, Hilbert point 13 decoded as 1011_2, which is collected as $x_0 = 11_2 = 3$, $x_1 = 01_2 = 1$, namely the point (3, 1) as shown in Fig. 5a. Figure 5a is a reasonable first step since most points already locate at the correct coordinates. Algorithm 2 shows the postprocess for Fig. 5a to transform wrong points into the desired coordinates in Fig. 5b.

We want to emphasize that the Hilbert curve provides only the **"locality"**. As shown in Fig. 6, we build a Hilbert curve with $n = 2, p = 5$ (i.e., two-dimensional space, $2^5 = 32$ discrete values for each dimension). The x-axis represents the Hilbert curve distance from the current point to the starting point; the y-axis represents the Euclidean distance from the current point to the starting point. We can observe that the Euclidean distance is not strictly monotone increasing but increase with fluctuations. In other words, a pair of

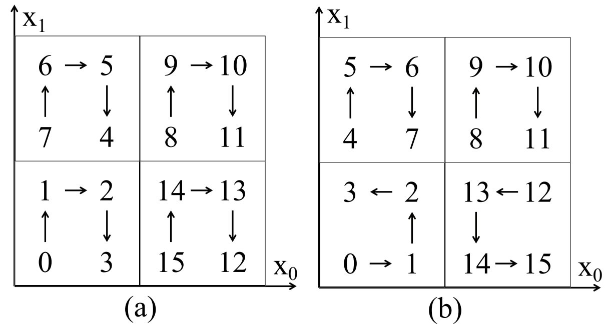

Fig. 5 An example of conversion from Hilbert curve distance to two-dimensional points. For example, for a Hilbert curve distance 13 whose binary representation is 1101_2, and $\lfloor 13/2 \rfloor$ is 6 whose binary representation is 0110_2. Then we calculate a initial position $1101_2 \oplus 0110_2 = 1011_2$ where $\oplus$ is a single exclusive-OR instruction. As shown in (**a**) part of this figure, $\underline{1}0\underline{1}1_2$ are collected as $x_0 = 11_2 = 3$ and $x_1 = 01_2 = 1$, i.e., (3, 1) is the initial position for 13. Algorithm 2 shows the postprocess for initial positions in (**a**) part of this figure, to transform wrong points into the desired Hilbert curve coordinates in (**b**) part of this figure

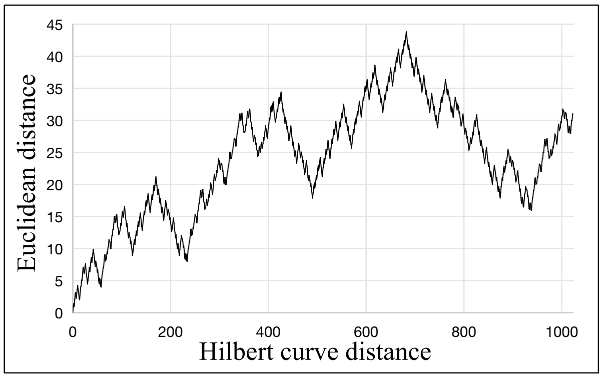

Fig. 6 We build a Hilbert curve with $n = 2, p = 5$ (i.e., two-dimensional space, $2^5 = 32$ discrete values for each dimension). We can observe that the Euclidean distance is not strictly monotone increasing but increase with fluctuations

Table 1 Detailed computation load analysis of Algorithm 4

Line	Computation load
1	(768 * **MUL** +768 * **ADD**)
2	C * (768 * **MUL** +768 * **ADD**)
3	768 * **MUL** + C * (768 * **MUL** +768 * **ADD**)
4	2 * C * **ADD**
5	C * **CMP**

embeddings are more probable (not absolute) has a smaller Euclidean distance when they own a smaller Hilbert curve distance.

Algorithm 2 Postprocess for Fig. 5(a)

```
1: for ($r = p-2, p-3, ..., 1, 0$):
2:     for ($i = n-1, n-2, ..., 1, 0$):
3:         if (bit $r$ of $x_i$ is 0):
4:             exchange low bits(r+1,r+2,...,p-1) of $x_i$ and $x_0$
5:         else:
6:             invert low bits of $x_0$
```

Algorithm 3 Constructing a Reordered Codebook

```
1: $[e^{d+k}] \leftarrow ([e^{d+k}] + 1)/stride$
2: $[distances] \leftarrow H([e^{d+k}])$
3: sorted, indices = sort([$distances$])
4: reordered_codebook = $[z^d][indices]$
5: for i in range(0, block_number):
6:     block_list[i]=reordered_codebook[$i \times b\_len : (i+1) \times b\_len$]
7:     block_mean_list[i]=mean(block_list[i])
8: Construct the power set $\mathcal{P}$ of block_list
```

We show how to construct a reordered codebook for VQ in Algorithm 3. Denote the original codebook as $[z^d]$ and the hyperbolic embedding enhanced codebook as $[e^{d+k}]$. Build a Hilbert curve function $H()$ that converts points to distance. Line 1 represents that we normalize the range of $[e^{d+k}]$ from $(-1, 1)$ to $(0, 2)$ and convert $[e^{d+k}]$ to points in n-dimensional hypercube $[0, 2^p)^n$ of side 2^p with unit precision, where $stride = 2/2^p$. Line 2 converts points to distance along the Hilbert curve. Lines 3–4 mean we sort the Hilbert curve distances and reorder the original codebook $[z^d]$ using the indices returned by the sort function. Lines 5–7 split the whole codebook into blocks and obtain the mean of each block, where b_len = **reordered_codebook**/$block\_number$. Line 8 constructs the power set of block_list.

3.3 Resizing codebook without retraining

When Algorithm 3 is finished, we obtain a codebook whose embeddings preserving the locality (to some extent, we can say it is "reordered" or "sorted"). Now we can resize the codebook of vector quantization without retraining. There are two typical demands from users: (1) using a smaller codebook for reducing computation load and energy consumption; (2) using a larger codebook for better reconstruction quality.

3.3.1 Smaller codebook

Algorithm 4 Pseudo Code of Vector Quantization

```
Input: z
Output: zq, indices
1: z_pow_2 = torch.sum(z ** 2, dim=1)
2: Codebook_pow_2 = torch.sum(Codebook**2, dim=1)
3: 2_z_Codebook = 2 * z @ Codebook
4: Euclidean_distance = z_pow_2 + Codebook_pow_2 + 2_z_Codebook
5: zq, indices = torch.min(Euclidean_distance, dim=1)
6: return zq, indices
```

Algorithm 4 shows a widely used[1] PyTorch-style pseudo code of VQ. Line 1 to 4 refer to a expansion that $(z - Codebook)^2 = z^2 + Codebook^2 - 2 * Codebook * z$We show a roughly computation load analysis of Algorithm 4 in Table 1. Where, we assume the shapes of z and Codebook are 1×768 and $C \times 768$, respectively. We use bold text to distinguish basic operators(MUL, ADD, CMP) from other parameters. We use bold text to distinguish basic operators (MUL, ADD, CMP) from other parameters. **MUL** is the multiplication operator, **ADD** is the addition operator, and **CMP** is the comparison operator between two numerical values.

We can observe that the computation load of Algorithm 4 depends on the codebook size C. If we can reduce C without a significant drop in reconstruction quality, we can reduce the computation load and energy consumption of VQ.

Searching the reordered codebook We let each query tensor z in a batch firstly finds the nearest mean tensor block_mean_list[j] and the corresponding candidate block block_list[j], as aforementioned in Algorithm 3. Then we dynamically use a smaller codebook (containing unique candidate blocks) for quantization. We denote this application as **HyperHill**. For example, we split the reordered codebook into B blocks, and an image is encoded as 256 query tensors. The best case is that these 256 query tensors have quantized tensors in one block, and we only need about $1/B$ time consumption compared to the original VQ. The worst case is that these 256 query tensors have quantized tensors which align uniformly in B blocks, and it falls back to the original VQ, which searches the whole codebook. In addition, manipulating subsets consisting of blocks introduces slow discontinuous memory access. Thus we build the power set of blocks containing 2^B subsets which are all contiguous tensors.

3.3.2 Larger codebook

For users who need better reconstruction quality, we can produce a larger codebook by conduct interpolation between adjacent embeddings in the codebook. We can uniformly insert $k - 1$ new embeddings between codebook entries e_i and e_{i+1} in Formula (13), We denote this application as **HyperHill-EXT**.

$$e_i + \frac{j}{k} * \left(e_{i+1} - e_i\right), j = 1, ..., k - 1 \tag{13}$$

4 Experiments

In this section, we evaluate HyperHill in various aspects. We also compare HyperHill against competitive baselines to show its effectiveness.

4.1 Experiment setup

4.1.1 Model settings

For Poincaré embeddings, the embedding dimension is 32. We train 500 epochs to obtain the embeddings with the learning rate to be 0.3. In the training process, we randomly sample 50 negative examples per positive example. For the Hilbert curve, we set $n = 256 + 32 = 288$ since z_q is 256-dimensional and θ is 32-dimensional. For all datasets, we set $p = 24$ to ensure that there is one point at a distance along the Hilbert curve on average.

In experiments, we use pre-trained VQGAN models[2] to reconstruct images in the validation datasets. We first report the averaged percentage of the used codebook in VQ. In addition, we quantify the degree of "realism" by computing FID [14] scores of reconstructed images (R-FIDs). We also evaluate the perceptual similarity between inputs and reconstructions with the LPIPS [40] metric and the structural similarity through PSNR [30] and SSIM [34]. All results are averaged over five trials on a server with Intel(R) Xeon(R) Platinum 8163 CPU and NVIDIA V100 32 G.

4.1.2 Datasets

We evaluate our method on following datasets. All RGB images are resized and center-cropped to 256×256, invalid images (i.e. not RGB, width<256, height<256) are

[1] https://github.com/CompVis/taming-transformers/blob/master/taming/modules/vqvae/quantize.py, https://github.com/MishaLaskin/vqvae/blob/master/models/quantizer.py.

[2] https://github.com/CompVis/taming-transformers.

Table 2 Statistics of each dataset

Dataset	Split	#Images
CelebAHQ	Train	25,000
	Validation	5000
	Test	5,000
COCO	Train	118,000
	Validation	5000
	Test	20,000
DIV2K	Train	800
	Validation	100
	Test	100

dropped. Experiments are conducted using the validation split of each dataset. Table 2 show statistics of each dataset used in our experiments.

1. CelebAHQ [17] is a higher-quality version of the CelebA dataset, which is a large-scale face attribute dataset with 40 attribute annotations [22]. We randomly select 25000/5000 images from CelebAHQ for training/validation;
2. COCO [21] is a large-scale object detection, segmentation, and captioning dataset. The main change of the dataset in 2017 is that instead of having an 83K/41K train/val split, the split is now 118K/5K for train/val based on community feedback. We use the new split in the experiments;
3. DIV2K [1, 15] is a large dataset of DIVerse 2K resolution images with a large diversity of contents. The dataset has 1000 images in total.

4.1.3 Baselines

VQ integrated with deep learning models (e.g., VQVAE and VQGAN) differs from VQ in domains, such as information retrieval and the database. VQ integrated with deep learning models is not only an approximate nearest neighbor search method, but its codebook is also a contiguous tensor which connects to the computational graph, supports gradient calculation, and conducts backpropagation. Thus, we choose Faiss k-means and LSH as baselines since they can keep the codebook as a contiguous tensor. Our **HyperHill**, Faiss k-means and LSH also do not occupy extra space for the index structure. It is an attractive feature since most GPUs have limited device memory. In future work, we will follow recent approximate nearest neighbor search methods and find effective and efficient codebook index strategies compatible with deep learning models.

- **Faiss** Faiss is a library for efficient similarity search and clustering of dense vectors. We leverage the Faiss k-means algorithm to build 8 clusters(train 20000 epochs). Each cluster centroid is assigned $codebook\_size/8$ nearest codebook entries.
- **Locality-Sensitive Hashing (LSH)** The problem of finding the nearest neighbors quickly in high-dimensional spaces can also be solved by LSH. LSH can make nearby vectors get the same hash with high probability, and distant ones do not. We realize the LSH baseline by employing random projections [19] as follows. Following the config in [19], we build 16 random embeddings that are 256-dimensional uniform distribution on the interval $(-1, 1)$. Then we sort the codebook entries by their sum of dot product with random embeddings and split the sorted codebook into 8 blocks.

Table 3 Overall results of **HyperHill** and baselines on CelebAHQ, COCO, and DIV2K

Dataset	Approach	Used codebook (%)↓	Time(s)	R-FID↓	LPIPS↓	PSNR↑	SSIM↑
CelebA	*Original*	*100.00*	10.41(100%)	*16.6711*	*0.1912*	*24.4743*	*0.6876*
HQ	Faiss	87.42	9.29(89.24%)	24.4032	0.2539	20.8086	0.6052
1024	LSH	78.23	8.44(81.08%)	16.6711	0.1912	24.4743	0.6876
	HyperHill	**60.36**	6.67(64.07%)	16.6482	0.1917	24.4451	0.6878
	Original	*100.00*	16.43(100%)	*19.0465*	*0.3301*	*17.4712*	*0.3989*
COCO	Faiss	78.96	13.39(81.49%)	32.7690	0.3870	17.4236	0.3381
8192	LSH	63.87	10.88(66.22%)	19.0465	0.3301	17.4712	0.3989
	HyperHill	**60.03**	10.24(62.32%)	19.0479	0.3301	17.4712	0.3989
	Original	*100.00*	0.92(100%)	*88.4467*	*0.3222*	*17.3721*	*0.4009*
DIV2K	Faiss	83.88	0.79(85.86%)	130.0410	0.4010	16.5645	0.3627
16384	LSH	64.13	0.63(68.47%)	88.4447	0.3221	17.3721	0.4009
	HyperHill	**52.00**	0.52(56.52%)	88.9347	0.3223	17.3716	0.4010

Table 4 Overall results of **HyperHill(VQVAE)** and baselines on CelebAHQ, COCO, and DIV2K

Dataset	Approach	Used codebook (%)↓	Time(s)	R-FID↓	LPIPS↓	PSNR↑	SSIM↑
CelebA	*Original*	*100.00*	4.11(100%)	*65.9795*	*0.5221*	*5.1383*	*0.2068*
HQ	Faiss	85.66	3.65(88.81%)	96.7264	0.6526	4.7612	0.1755
1024	LSH	68.72	2.94(71.53%)	65.9421	0.5224	5.1295	0.2066
	HyperHill (VQVAE)	**63.19**	2.77(67.39%)	65.9733	0.5229	5.1291	0.2064
	Original	*100.00*	6.35(100%)	*64.8080*	*0.5586*	*5.0303*	*0.1960*
COCO	Faiss	83.82	5.54(87.24%)	98.6372	0.6831	4.2674	0.1668
8192	LSH	65.49	4.36(68.66%)	64.7973	0.5589	5.0301	0.1957
	HyperHill (VQVAE)	**61.74**	4.09(64.41%)	64.8218	0.5590	5.0298	0.1955
	Original	*100.00*	0.34(100%)	*208.3688*	*0.5533*	*4.9660*	*0.1772*
DIV2K	Faiss	86.64	0.31(91.18%)	295.8846	0.6924	4.1187	0.1525
16384	LSH	69.56	0.25(73.53%)	208.3359	0.5537	4.9655	0.1769
	HyperHill (VQVAE)	**58.31**	0.22(64.71%)	208.5842	0.5540	4.9654	0.1766

Table 5 Metrics for different block numbers when we reconstruct DIV2K using VQGAN pre-trined on ImageNet

# Blocks	*1*	4	6	8	10
R-FID↓	*88.45*	88.70	88.48	88.95	88.57
LPIPS↓	*0.32*	0.32	0.32	0.32	0.32
PSNR↑	*17.37*	17.37	17.38	17.37	17.38
SSIM↑	*0.40*	0.40	0.40	0.40	0.40
Used Codebook(%)↓	*100*	80.75	58.00	51.63	42.90
Space	*1x*	8x	32x	128x	512x

4.2 Overall results

Table 3 shows the overall results of **HyperHill** and baselines on three datasets. The number under the dataset name represents the codebook size of the corresponding VQGAN model. It will show the robustness of **HyperHill** for different codebook sizes of different datasets. The original performance of VQGAN is italic and bold values indicate the smallest used codebook percentage. To make a fair comparison, for Faiss and LSH, we use the same query method as **HyperHill**, described in Sect. 3.3.1. The **Time(s)** column reports the total time spent on the VQ process of Original VQGAN, Faiss, LSH, and HyperHill. Numbers (in parentheses) after spent time is the percentage of the time spent by the Original VQ process. From Table 3, we have following observations.

- Faiss needs to use 80–90% codebook size but obtains poorer behavior on metrics than the original VQ.
- LSH can achieve similar performance compared to the original VQ, but LSH uses random projection so that its behavior is not stable. In addition, the used codebook for LSH varies from 63.87 to 78.23%, the average of the used codebook is about 11.28% more than the average of **HyperHill**.
- The proposed **HyperHill** stably uses 50% to 60% codebook and obtains similar metrics compared to the original VQ. As aforementioned, **HyperHill** provides other significant features, such as needing no training, producing consistent results, and keeping the codebook contiguous.
- The percentage of spent time is about 3% higher than the percentage of used codebook since we need to find candidate blocks for each query tensor which needs quantization, as aforementioned.

In addition, we do not take the computation cost of training the hyperbolic embedding and Hilbert codebook reordering into consideration since they only need to be done offline once.

Table 6 Ablation study on CelebAHQ, COCO and DIV2K. "w/o. HE" refers to our approach without hyperbolic embeddings

Dataset	Approach	Used Codebook (%)↓	R-FID↓	LPIPS↓
CelebAHQ	HyperHill	60.36	16.6482	0.1917
	w/o. HE	65.21	16.6564	0.1914
COCO	HyperHill	60.03	19.0479	0.3301
	w/o. HE	69.72	19.0490	0.3301
DIV2K	HyperHill	52.00	88.9347	0.3223
	w/o. HE	62.50	88.6303	0.3222

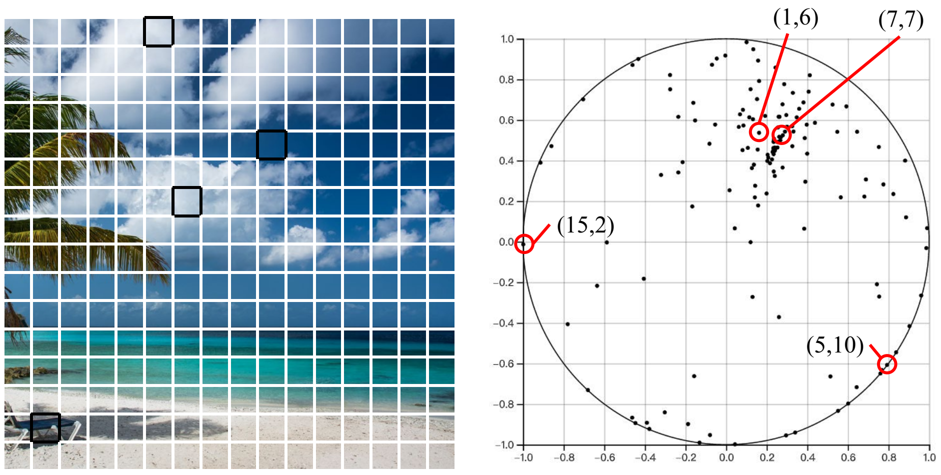

Fig. 7 two-dimensional Poincaré embeddings of Fig. 2. Patches are converted to VQ indices. Indices that are probable to co-occur in a batch get closer to each other in the Euclidean space. However, only 204 out of 256 image tokens have appeared in the training split of the DIV2K dataset, so they own two-dimensional Poincaré embeddings

To test whether our **HyperHill** is compatible with other VQ-inside deep learning models, we build a variant **HyperHill(VQVAE)** whose VQGAN models are replaced by a VQVAE[3] model (with embedding dimension is 256, and codebook size is 8192), pre-trained by Ding et al. [7]. As shown in Table 4, the original performance of VQVAE is italic and bold values indicate the smallest used codebook percentage, **HyperHill(VQVAE)** also can achieve similar performance to the original VQVAE. **HyperHill(VQVAE)** utilize averagely about 60% codebook and time consumption, outperforming Faiss and LSH. In the rest of the experiments, we focus on **HyperHill** (using VQGAN) rather than **HyperHill(VQVAE)** since VQVAE produces poor metrics (e.g., R-FID, LPIPS, PSNR, and SSIM) in reconstructing images.

4.3 The effects of block numbers

Table 5 shows the metrics for different block numbers when we reconstruct DIV2K using VQGAN. Italic metrics of block number 1 represent the original performance of VQGAN. We can observe that 8 blocks use about half (51.63%) codebook size. While the codebook size is 16384, it will occupy extra

$$32bit \times 256 \times 16,384 \times 128 = 2\,GB \tag{14}$$

memory for the power set

$$\mathcal{P}$$

in Algorithm 3. Thus, we should select the proper block number according to available hardware memory. For experiments in this paper, we set the block number to 8 as it provides a relatively good tradeoff between computation efficiency and memory consumption.

4.4 The effects of hyperbolic embeddings

As shown in Table 6, **HyperHill** without hyperbolic embeddings requires a higher proportion of the codebook entries (i.e., 4.85%, 9.69%, and 10.5%) to obtain similar metrics compared to the full **HyperHill** implementation. It proves that the hyperbolic embeddings are effective for capturing the co-occurrence information and the logical

[3] https://openi.pcl.ac.cn/BAAI/WuDao-Model/src/branch/master/CogView.

Table 7 Overall results of **HyperHill-EXT** codebook interpolation. Bold numbers are best values for corresponding metrics

Dataset	Approach	Codebook size	R-FID↓	LPIPS↓	PSNR↑	SSIM↑
	Original	*1024*	***16.6711***	*0.1912*	*24.4743*	*0.6876*
CelebAHQ	$k = 2$	2047	16.9779	0.1891	24.5716	0.6902
1024	$k = 4$	4093	17.1357	0.1877	24.6257	0.6916
	$k = 8$	8185	17.1684	0.1872	24.6441	0.6921
	$k = 16$	16369	17.1659	**0.1871**	**24.6493**	**0.6923**
	Original	*8192*	*19.0465*	*0.3301*	*17.4712*	*0.3989*
COCO	$k = 2$	16383	18.7634	0.3256	17.5901	0.4062
8192	$k = 4$	32765	**18.6959**	0.3234	17.6498	0.4098
	k=8	65529	18.6983	0.3226	17.6726	0.4113
	$k = 16$	131057	18.7043	**0.3224**	**17.6809**	**0.4119**
	Original	*16384*	*88.4467*	*0.3222*	*17.3721*	*0.4009*
DIV2K	$k = 2$	32767	**87.3981**	0.3180	**17.3769**	0.4046
16384	$k = 4$	65533	90.7893	0.3168	17.3626	0.4065
	$k = 8$	131065	89.6003	0.3164	17.3583	0.4073
	$k = 16$	262129	89.2086	**0.3162**	17.3568	**0.4076**

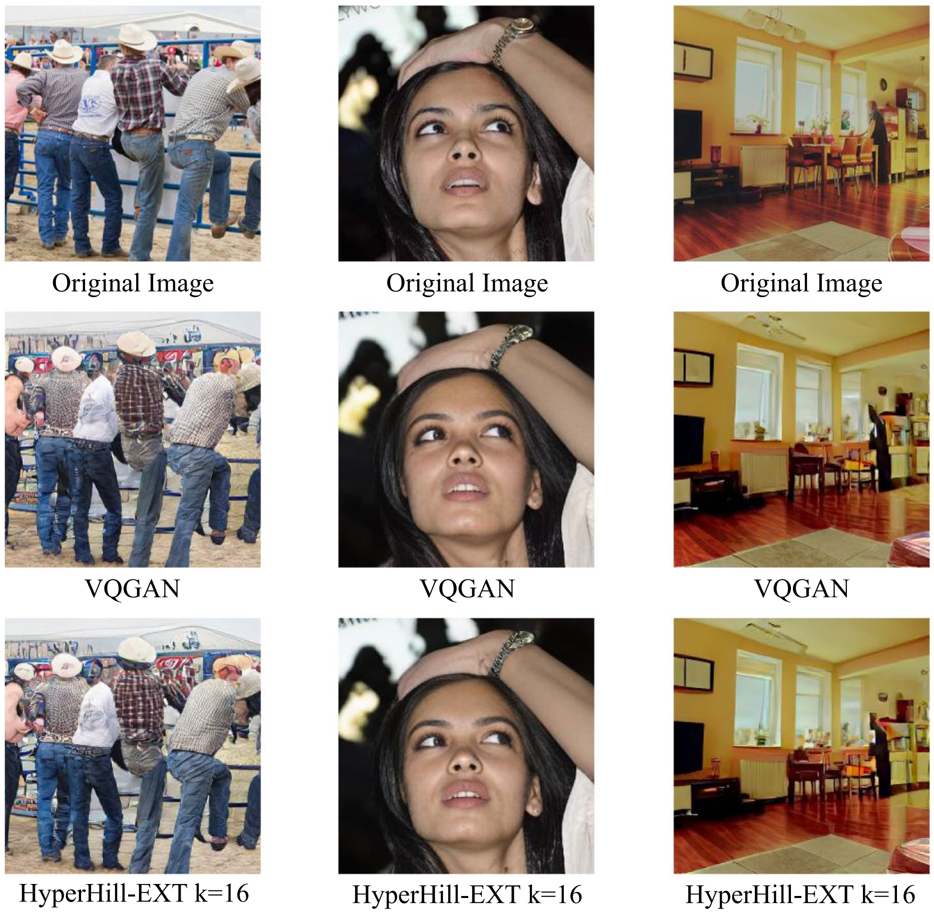

Fig. 8 Comparison of reconstruct quality of VQGAN and HyperHill-EXT with $k = 16$. The first column is from the DIV2K dataset. The second column is from the CelebAHQ dataset. The third column is from the COCO dataset

similarity between image patches. It should be further noted that unseen image tokens that do not own their hyperbolic embeddings are represented by the uniform distribution in the interval $(-1, 1)$. The presence of unseen image tokens may hurt the performance of hyperbolic embeddings.

4.5 Case study

As shown in Fig. 7, we present the two-dimensional Poincaré embeddings of Fig. 2. The original image in Fig. 2 is resized to 256×256 and is mapped to latent embeddings of the size 16×16=256 after four downsampling steps. After VQ, we obtain 256 discrete image tokens (codebook indices), together with their two-dimensional Poincaré embeddings. We can observe the clustering of image tokens, and 74% of the image tokens lie in the first quadrant. We also present positions of four patches labeled as (row, col). These cases empirically prove that Poincaré embeddings can make frequently co-occurred image tokens get closer in the Euclidean space.

4.6 Interpolation results for HyperHill-EXT

In Table 7, we present the overall results of **HyperHill-EXT** codebook interpolation. Italic values represent that we use original codebooksize. Bold values indicate the best metrics. The parameter k means taht we uniformly insert $k-1$ new embeddings between codebook entries e_i and e_{i+1}, as aforementioned in Formula (13). We have following observations:

- We can obtain better R-FID and PSNR by changing the codebook size.
- The largest codebook size produces best LPIPS and SSIM.
- **HyperHill-EXT** needs no retraining, but it is compatible with retraining. Based on **HyperHill-EXT**, users can find the best codebook size for their own datasets. Then they can retrain(fine-tune) VQGAN using the best codebook size.

Furthermore, as shown in Fig. 8,we provide a comparison of reconstruct quality of VQGAN and HyperHill-EXT with $k = 16$. We can roughly observe that HyperHill-EXT with $k = 16$ offer better reconstruction quality, such as “hats” in the first column, “eyelashes” in the second column, and “carpets” in the third column. Figure 8 supports conclusions that we drawn from Table 7.

5 Conclusion and future work

To our best knowledge, HyperHill is a novel approach to resize codebook of vector quantization without retraining. We firstly employ hyperbolic embeddings to enhance codebook entries with logical similarities. Then, we can utilize the Hilbert curve to produce reasonable and stable codebook splits than baselines, e.g., Faiss and LSH. For users preferring a smaller codebook and lower computation load, **HyperHill** can use a smaller subset of the codebook to achieve similar metrics compared to the original VQ. For users who require better reconstruction quality, **HyperHill-EXT** can improve R-FID, LPIPS, PSNR, and SSIM by interpolations between codebook entries. Experimental results show the effectiveness of HyperHill against competitive baselines. In the future, we will provide efficient CUDA implementations of HyperHill and integrate more indexing and partitioning strategies from vector databases, e.g., Milvus and Faiss.

Acknowledgements This work has been supported by the National Natural Science Foundation of China under Grant No. U1911203, 61877018, 61977025, 62202170, and Alibaba Group through the Alibaba Innovation Research Program.

Author Contributions LL and TL proposed main ideas of this paper and conducted all experiments. LL, TL, CW, and MQ wrote the main manuscript text. CC, MG, and AZ provided computing resource for experiments and reviewed the manuscript

Funding National Natural Science Foundation of China under Grant No. U1911203 National Natural Science Foundation of China under Grant No. 61877018 National Natural Science Foundation of China under Grant No. 61977025 National Natural Science Foundation of China under Grant No. 62202170 Alibaba Innovation Research Program

Availability of supporting data All datasets and model weights used in our experiments are publicly available with references or URLs. Our proposed approaches are in detail described with pseudo codes or figures. In addition, we will release all codes as a open-source project upon acceptance of this paper.

Declarations

Conflict of interest All authors declare that the authors have no competing interests as definedby Springer, or other interests that might be perceived to influence the resultsand/or discussion reported in this paper.

Ethical approval and consent to participate Not applicable.

Consent for publication We would like to declare that this manuscript has not been published previously and is not under consideration for any other conferences or journals. All the authors have approved the manuscript for publication.

Human and animal ethics Not applicable.

References

1. Agustsson, E., Timofte, R.: NTIRE 2017 challenge on single image super-resolution: dataset and study. In: Proceedings of the IEEEconference on computer vision and pattern recognition workshops, pp. 1122–1131 (2017)

2. Ai, L., Yu, J., Wu, Z., et al.: Optimized residual vector quantization for efficient approximate nearest neighbor search. Multimed. Syst. **23**(2), 169–181 (2017)
3. Bai, Y., Ying, Z., Ren, H., et al.: Modeling heterogeneous hierarchies with relation-specific hyperbolic cones. IIn: Advances in NeuralInformation Processing Systems, pp. 12316–12327 (2021)
4. Chen, H., Chang, Y.: All-nearest-neighbors finding based on the Hilbert curve. Expert Syst. Appl. **38**(6), 7462–7475 (2011)
5. Devlin, J., Chang, M., et al.: BERT: pre-training of deep bidirectional transformers for language understanding. In: arXiv preprintarXiv: https://arxiv.org/1810.04805 (2019
6. Ding, M., Kong, K., Li, J., et al.: Vq-gnn: a universal framework to scale up graph neural networks using vector quantization. arXiv: https://arxiv.org/2110.14363 (2021)
7. Ding, M., Yang, Z., et al.: Cogview: mastering text-to-image generation via transformers. CoRR arXiv: https://arxiv.org/abs/2105.13290 (2021)
8. Esser, P., Rombach, R., et al.: Taming transformers for high-resolution image synthesis. In: Proceedings of the IEEE/CVF conference oncomputer vision and pattern recognition, pp. 12873–12883 (2021)
9. Fei, N., Lu, Z., et al.: Wenlan 2.0: make AI imagine via a multimodal foundation model. CoRR arXiv:https://arxiv.org/abs/2110.14378 (2021)
10. Fu, C., Xiang, C., Wang, C., et al.: Fast approximate nearest neighbor search with the navigating spreading-out graph. Proc. VLDB Endow. **12**(5), 461–474 (2019)
11. Ge, T., He, K., Ke, Q., et al.: Optimized product quantization for approximate nearest neighbor search. In: Proceedings of the IEEEConference on Computer Vision and Pattern Recognition, pp. 2946–2953 (2013)
12. Guo, R., Sun, P., Lindgren, E., et al.: Accelerating large-scale inference with anisotropic vector quantization.In: International-Conference on Machine Learning, vol 119. PMLR, pp. 3887–3896 (2020)
13. He, K., Wen, F., Sun, J.: K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In:Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2938–2945 (2013)
14. Heusel, M., Ramsauer, H., Unterthiner, T., et al.: Gans trained by a two time-scale update rule converge to a local nash equilibrium.In:Advances in neural information processing systems, pp. 6626–6637 (2017)
15. Ignatov, A., Timofte, R., et al.: PIRM challenge on perceptual image enhancement on smartphones: report. In: Proceedings of theEuropean Conference on Computer Vision (ECCV) Workshops, pp. 315–333 (2018)
16. Kalantidis, Y., Avrithis, Y.: Locally optimized product quantization for approximate nearest neighbor search. In: Proceedings of theIEEE Conference on Computer Vision and Pattern Recognition, pp. 2329–2336 (2014)
17. Karras, T., Aila, T., et al.: Progressive growing of gans for improved quality, stability, and variation. In: ICLR (2018)
18. Khrulkov, V., Mirvakhabova, L., et al.: Hyperbolic image embeddings. n: Proceedings of the IEEE/CVF Conference on Computer-Vision and Pattern Recognition, pp. 6417–6427 (2020)
19. Kitaev, N., Kaiser, L., et al.: Reformer: the efficient transformer. In: ICLR (2020)
20. Lin, J., Men, R., et al.: M6: A chinese multimodal pretrainer. CoRR arXiv: https://arxiv.org/abs/2103.00823 (2021)
21. Lin, T., Maire, M., et al.: Microsoft COCO: common objects in context.In: Proceedings of 13th European Conference of ComputerVision (ECCV), pp. 740–755 (2014)
22. Liu, Z., Luo, P., et al.: Deep learning face attributes in the wild. In: Proceedings of the IEEE international conference on computer vision, pp. 3730–3738 (2015)
23. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans. Pattern Anal. Mach. Intell. **42**(4), 824–836 (2020)
24. Nickel, M., Kiela, D.: Poincaré embeddings for learning hierarchical representations. In: Advances in neural information processingsystems, pp. 6338–6347 (2017)
25. van den Oord, A., Vinyals, O., et al.: Neural discrete representation learning. In: Advances in neural information processing systems, pp. 6306–6315 (2017)
26. Peng, W., Varanka, T., et al.: Hyperbolic deep neural networks: a survey. CoRR arXiv: https://arxiv.org/abs/2101.04562 (2021)
27. Radford, A., Wu, J., Child, R., et al.: Language models are unsupervised multitask learners. CoRR (2019)
28. Razavi, A., van den Oord, A., et al.: Generating diverse high-fidelity images with VQ-VAE-2. In: Advances in neural informationprocessing systems, pp. 14837–14847 (2019)
29. Roy, A., Grangier, D.: Unsupervised paraphrasing without translation. In: arXiv preprint arXiv: https://arxiv.org/1905.12752 (2019)
30. Setiadi, D.R.I.M.: PSNR vs SSIM: imperceptibility quality assessment for image steganography. Multimed. Tools Appl. **80**(6), 8423–8444 (2021)
31. Skilling, J.: Programming the Hilbert curve. In: Bayesian Inference and Maximum Entropy Methods in Science and Engineering, pp. 381–387 (2004). https://aip.scitation.org/doi/abs/10.1063/1.1751381. https://aip.scitation.org/toc/apc/707/1. https://www.aip.org/aip/history/locations
32. Tsinganos, P., Cornelis, B., et al.: A Hilbert curve based representation of semg signals for gesture recognition. In: International Conference on Systems, Signals and Image Processing (IWSSIP), pp. 201–206 (2019)
33. Wang, L., Hu, F., Wu, S., et al.: Fully hyperbolic graph convolution network for recommendation. In: Proceedings of the 30th ACM10.1007/s00530-023-01065-2International Conference on Information & Knowledge Management, pp. 3483–3487 (2021)
34. Wang, Z., Bovik, A.C., Sheikh, H.R., et al.: Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. **13**(4), 600–612 (2004)
35. Wu, C., Liang, J., et al.: Nüwa: visual synthesis pre-training for neural visual world creation.CoRR arXiv: https://arxiv.org/abs/2111.12417 (2021)
36. Wu, X., Guo, R., Suresh, A.T., et al.: Multiscale quantization for fast similarity search. In: Advances in neural information processingsystems, **30** pp. 5745–5755 (2017)
37. Wu, Y., Cao, X., An, Z.: A spatiotemporal trajectory data index based on the Hilbert curve code. In: IOP Conference Series: Earth andEnvironmental Science, Vol. 502, No. 1, pp. 012005 (2020)
38. Yang, M., Zhou, M., Liu, J., et al.: HRCF: enhancing collaborative filtering via hyperbolic geometric regularization.In: Proceedings ofthe ACM Web Conference, pp. 2462–2471 (2022)
39. Zhang, H., Yin, W., et al.: Ernie-vilg: Unified generative pre-training for bidirectional vision-language generation. CoRR arXiv: https://arxiv.org/abs/2112.15283 (2021)
40. Zhang, R., Isola, P., Efros, A.A., et al.: The unreasonable effectiveness of deep features as a perceptual metric. In: CVPR. Computer Vision Foundation/IEEE Computer Society, pp. 586–595 (2018)