# PQk-means: Billion-scale Clustering for Product-quantized Codes

Yusuke Matsui*
National Institute of Informatics, Japan
matsui@nii.ac.jp

Keisuke Ogaki*
DWANGO Co., Ltd., Japan
keisuke_ogaki@dwango.co.jp

Toshihiko Yamasaki
The University of Tokyo, Japan
yamasaki@hal.t.u-tokyo.ac.jp

Kiyoharu Aizawa
The University of Tokyo, Japan
aizawa@hal.t.u-tokyo.ac.jp

## ABSTRACT

Data clustering is a fundamental operation in data analysis. For handling large-scale data, the standard k-means clustering method is not only slow, but also memory-inefficient. We propose an efficient clustering method for billion-scale feature vectors, called *PQk-means*. By first compressing input vectors into short product-quantized (PQ) codes, PQk-means achieves fast and memory-efficient clustering, even for high-dimensional vectors. Similar to k-means, PQk-means repeats the assignment and update steps, both of which can be performed in the PQ-code domain. Experimental results show that even short-length (32 bit) PQ-codes can produce competitive results compared with k-means. This result is of practical importance for clustering in memory-restricted environments. Using the proposed PQk-means scheme, the clustering of one billion 128D SIFT features with $K = 10^5$ is achieved within 14 hours, using just 32 GB of memory consumption on a single computer.

## CCS CONCEPTS

• **Information systems** → *Clustering*; *Nearest-neighbor search*; • **Theory of computation** → *Unsupervised learning and clustering*;

## KEYWORDS

k-means; clustering; product quantization; billion-scale

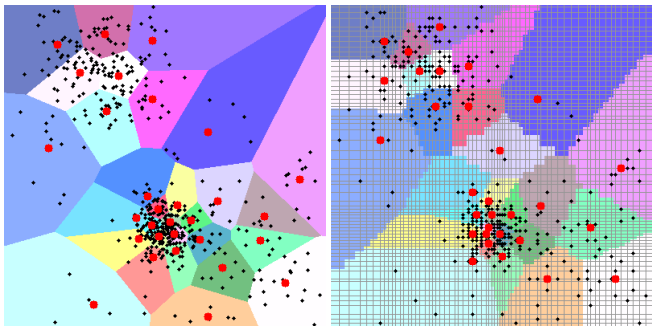
## 1 INTRODUCTION

Many recent advances in computer vision are attributed to supervised learning with several annotated data sources. However, manual annotation is a time-consuming and laborious task. Clustering (unsupervised learning) is a promising method for taking better advantage of unlabeled data [41]. Specifically, we focus on million- or billion-scale clustering for data with hundreds or thousands of dimensions, e.g., clustering on 100 million images with 4096D AlexNet features (YFCC100M [36]).

*Joint first authors.


*MM '17, October 23–27, 2017, Mountain View, CA, USA*

ACM ISBN 978-1-4503-4906-2/17/10…$15.00
https://doi.org/10.1145/3123266.3123430



(a) k-means (b) PQk-means

**Figure 1: A 2D example using both k-means and PQk-means, with $K = 30$. (a) K-means applied to 500 2D vectors (black dots), and the resulting 30 centers denoted as red circles. (b) The same 500 vectors encoded as 500 PQ codes (black dots), and the resulting 30 centers using PQk-means. Because both dimensions ($x$ and $y$) are quantized independently, the PQ codes are placed on nonuniformly quantized lattices.**

The problems of large-scale clustering include large memory consumption and prohibitive runtime costs. Owing to these two issues, the standard k-means clustering method [22] can barely handle large-scale data. Distributed batch clustering [26, 34] is a possible solution for achieving large-scale clustering within a reasonable timescale. However, this requires vast computational resources. For example, clustering 100 million features within several hours requires 300 machines [2] in a Spark framework[1].

In this paper, we propose *PQk-means*, which is a **billion-scale** clustering method, and can be performed on a **single computer** with only a **reasonable memory consumption** (less than 32 GB of RAM) **within a single day**. The key idea is to first compress input vectors into memory-efficient short codes by product quantization [18], and to then cluster the resultant product-quantized (PQ) codes (rather than the original vectors) in the compressed domain. As with k-means, PQk-means also repeats the following two steps until convergence is achieved: (1) Find the nearest center from each code, and (2) update each center using a proposed *sparse voting* scheme.

[1] http://spark.apache.org/

Fig. 1 illustrates a 2D example, comparing k-means with PQk-means. The result of the PQk-means procedure is similar to that of k-means, although PQk-means is 5.3 times more memory efficient[2].

The technical challenge lies in the direct computation of the center of a new cluster. Because PQ codes consist exclusively of sets of identifiers (integers), averaging operations on such codes cannot be explicitly defined. A naïve brute-force updating method is slow, because all possible candidates must be evaluated. To solve this problem, we develop an alternative fast method, called *sparse voting*. We consider a frequency histogram of the assigned PQ codes in each cluster. Owing to the nature of clustering, this histogram is usually sparse. By focusing only on non-zero elements in the histogram, we can omit most calculations. Sparse voting is a simple procedure. However, it significantly accelerates the computation, and achieves exactly the same results as the naïve method.

We analyzed the runtime and memory consumption of PQk-means. Moreover, we intensively compared PQk-means with existing methods, such as k-means [22], Bk-means [14], Ak-means [31], and IQ-means [2], using the SIFT1M and ILSVRC_1000C datasets. Billion-scale evaluation was also investigated, using the YFCC100M, SIFT1B, and Deep1B datasets.

The contributions of this paper are as follows:

- We develop PQk-means, a billion-scale memory-efficient clustering algorithm. The clustering of one billion 128D SIFT vectors with $K = 10^5$ was achieved in 14 hours, using just 32 GB of RAM. Note that standard k-means clustering requires 512 GB of RAM just to represent the data.
- PQk-means is conceptually simple, and straightforward to implement.
- Unlike existing large-scale clustering methods such as Bk-means [14] or IQ-means [2], the original vectors can be approximately reconstructed following the clustering. This is a useful property if the original vectors are required after clustering.
- Experimental results show that clustering with short-length PQ codes (e.g., 32 bit) is still effective (see Fig. 5 for visual examples). This is a practically important result for memory-efficient clustering.

## 2 RELATED WORK

Data clustering is a fundamental operation in data analysis [15, 17]. Since the original k-means clustering method was proposed [22], several theoretical improvements have been presented. In particular, the provision of good seeds [1, 6, 7] and bounding-based acceleration [11, 25] are still being intensively studied. Because these algorithms are based on k-means, they always produce the same final clustering result with the same initial seed [25].

Considering real-world applications, faster algorithms are in high demand, even though the result of clustering in such cases is only an approximation of that of k-means. Such approximated k-means methods include approximated search [31], hierarchical search [27], approximated bounds [38], and batch-based methods [26, 34]. If the size of the input data is large, subset-based methods [2, 8] can achieve the fastest performance. These methods only treat a subset of the input vectors (i.e., vectors close to each center), making the computation efficient. The current state-of-the-art for subset-based methods is IQ-means [2]. While subset-based methods are fast, their accuracy is not always competitive compared with other methods, because only subsets of vectors are used.

For handling large-scale data (e.g., $10^8$ 4096D AlexNet features, which require 1.6 TB in total using `float`), memory consumption also constitute an important issue. Binary k-means (Bk-means) [14] converts input vectors into binary codes [9, 13], so that all of the binary codes can be stored in memory. The Hamming distance between two binary codes approximates the Euclidean distance between their original vectors. Because the Hamming distance can be computed efficiently by either a linear scan or hash table [29, 30], Bk-means achieves fast clustering with efficient memory utilization. The drawbacks of Bk-means are two-fold. First, binary conversion is less accurate than quantization-based compression [3, 4, 10, 12, 28, 39, 42, 43], as has been discussed in the nearest neighbor search community [40]. Second, we cannot reconstruct the original vectors from the resultant binary codes. Our PQk-means method addresses these two concerns. Experimental results show that PQk-means always achieves a better accuracy than Bk-means with the same code length. A comparison of standard k-means, Bk-means, and PQk-means is summarized in Table 1.

## 3 BACKGROUND

In this section, we briefly review k-means [22] for clustering, and product quantization [18] for encoding.

### 3.1 K-means clustering

The k-means algorithm finds $K$ cluster centers such that the sum of the distances between each vector and its closest center is minimized. Specifically, given $N$ $D$-dimensional vectors $\mathcal{X} = \{\boldsymbol{x}_n \in \mathbb{R}^D\}_{n=1}^N$, one must find $K$ centers $\{\boldsymbol{\mu}_k \in \mathbb{R}^D\}_{k=1}^K$ that minimize the following cost function [22, 26]:

$$E\left(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\right) = \frac{1}{N} \sum_{n=1}^{N} d\left(\boldsymbol{x}_n, \boldsymbol{\mu}_{a(n)}\right), \tag{1}$$

where $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2$. Note that $a(n)$ is an assignment function, defined by $a(n) = \arg\min_{k \in \{1,\ldots,K\}} d(\boldsymbol{x}_n, \boldsymbol{\mu}_k)^2$.

The cost function converges to a local minimum by repeating the following two steps. In the **assignment step**, each vector is assigned to the nearest center. This is achieved by computing $a(n)$ for each $n \in \{1, \ldots, N\}$. In the **update step**, each center is updated by averaging over the assigned vectors, $\boldsymbol{\mu}_k \leftarrow \frac{1}{|\mathcal{X}_k|} \sum_{\boldsymbol{x} \in \mathcal{X}_k} \boldsymbol{x}$, where $\mathcal{X}_k = \{\boldsymbol{x}_n \in \mathcal{X} | a(n) = k\}$.

### 3.2 Product quantization for encoding

The product-quantization algorithm encodes input vectors into short codes [18]. A $D$-dimensional input vector $\boldsymbol{x} \in \mathbb{R}^D$ is split into $M$ disjointed subvectors. For each $D/M$-dimensional subvector, the closest codeword from the pre-trained $L$ codewords is determined, and its index (an integer in $\{1, 2, \ldots, L\}$) is recorded. Finally, $\boldsymbol{x}$ is encoded as $\bar{\boldsymbol{x}}$, which is a tuple of $M$ integers defined as follows:

$$\boldsymbol{x} \mapsto \bar{\boldsymbol{x}} = [\bar{x}^1, \ldots, \bar{x}^M]^\top \in \{1, \ldots, L\}^M, \tag{2}$$

where the $m$th subvector in $\boldsymbol{x}$ is quantized into $\bar{x}^m$. We refer to $\bar{\boldsymbol{x}}$ as a PQ code for $\boldsymbol{x}$. Note that $\bar{\boldsymbol{x}}$ is represented by $M \log_2 L$ bits. We

[2]For representing each dimension, a 32 bit `float` is used in k-means, and a 6 bit integer is used in PQk-means ($2^6 = 64$ codewords are used).

**Table 1: Comparison of k-means, Bk-means, and the proposed PQk-means clustering methods. In k-means clustering, the vector is a $D$-dimensional real-valued vector. In Bk-means clustering, the vector is a $B$-bit binary string. In PQk-means clustering, the vector is a tuple consisting of $M$ indices, whose range is from 1 to $L$.**

| Method | Representation | Step1: Assignment | Step2: Updating |
|---|---|---|---|
| k-means [22] | $\boldsymbol{x} \in \mathbb{R}^D$ | Nearest neighbor search | Averaging |
| Bk-means [14] | $\boldsymbol{x} \mapsto \boldsymbol{x}_b \in \{0,1\}^B$ | Hash table for binary codes [29] | Bit operation [14] |
| PQk-means (proposed) | $\boldsymbol{x} \mapsto \bar{\boldsymbol{x}} \in \{1,\dots,L\}^M$ | Hash table for PQ codes [23] (Sec. 4.1) | Sparse voting (Sec. 4.2) |

set $L$ to 256, in order to represent each code using $M$ bytes. This is a typical setting in many studies.

Note that for each subspace, $L$ codewords are trained beforehand. Therefore, we can compute a distance matrix among codewords for each subspace, $A^m \in \mathbb{R}^{L\times L}$ for each $m \in \{1,\dots,M\}$, where $A^m_{i,j}$ denotes the squared distance between the $i$th and $j$th codewords for the $m$th subspace.

Suppose we have two vectors, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, and that their PQ codes are $\bar{\boldsymbol{x}}_1$ and $\bar{\boldsymbol{x}}_2$, respectively. Then, the Euclidean distance between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is efficiently approximated with the two codes $\bar{\boldsymbol{x}}_1$ and $\bar{\boldsymbol{x}}_2$; $d(\boldsymbol{x}_1,\boldsymbol{x}_2) \sim d_{SD}(\bar{\boldsymbol{x}}_1,\bar{\boldsymbol{x}}_2)$. This is known as the symmetric distance (SD) [18]:

$$d_{SD}(\bar{\boldsymbol{x}}_1,\bar{\boldsymbol{x}}_2)^2 = \sum_{m=1}^{M} d^m_{SD}(\bar{x}^m_1,\bar{x}^m_2)^2 = \sum_{m=1}^{M} A^m_{\bar{x}^m_1,\bar{x}^m_2}. \tag{3}$$

The SD approximates the distance between the original vectors by the distance between codewords, denoted by PQ codes. Here, $d^m_{SD}(i,j)$ computes the distance between the $i$th and $j$th codewords in the $m$th space, and can be computed simply by looking up $A^m_{i,j}$. Therefore, the squared SD can be efficiently computed using look-up tables with a summation of the results. This computation requires a cost of $O(M)$.

A useful property of product quantization is its reconstructability. Given a PQ code $\bar{\boldsymbol{x}}$, an original vector $\boldsymbol{x} \in \mathbb{R}^D$ can be approximately reconstructed by fetching the codewords $\bar{\boldsymbol{x}} \mapsto \hat{\boldsymbol{x}} \in \mathbb{R}^D$, where $\hat{\boldsymbol{x}}$ is an approximation of $\boldsymbol{x}$.

## 4 PQK-MEANS CLUSTERING

In this section, we present our proposed PQk-means clustering method. We assume that $D$-dimensional input vectors $\mathcal{X} = \{\boldsymbol{x}_n\}_{n=1}^N$ are encoded beforehand using product quantization, as $\bar{\mathcal{X}} = \{\bar{\boldsymbol{x}}_n\}_{n=1}^N$. Our objective is to determine $K$ cluster centers that minimize the cost function:

$$E(\bar{\boldsymbol{\mu}}_1,\dots,\bar{\boldsymbol{\mu}}_K) = \frac{1}{N}\sum_{n=1}^{N} d_{SD}\left(\bar{\boldsymbol{x}}_n, \bar{\boldsymbol{\mu}}_{a(n)}\right), \tag{4}$$

where $\bar{\boldsymbol{x}}_n = [\bar{x}^1_n,\dots,\bar{x}^M_n]^\top \in \{1,\dots,L\}^M$. Note that each cluster center $\bar{\boldsymbol{\mu}}_k = [\bar{\mu}^1_k,\dots,\bar{\mu}^M_k]^\top \in \{1,\dots,L\}^M$ is also a PQ code. Here, Eq. (4) differs from Eq. (1) in two aspects. First, both input vectors and centers are PQ codes. Second, the symmetric distance $d_{SD}$ is used to measure the distance between two PQ codes.

Similar to the standard k-means clustering method, PQk-means repeats the assignment and update steps until convergence is achieved.

### 4.1 Assignment step

In the assignment step, the nearest center in terms of the SD is determined for each $\bar{\boldsymbol{x}}_n$:

$$a(n) = \underset{k\in\{1,\dots,K\}}{\arg\min}\ d_{SD}(\bar{\boldsymbol{x}}_n,\bar{\boldsymbol{\mu}}_k)^2. \tag{5}$$

There are two methods to compute Eq. (5): the PQ linear scan [18] or PQTable [23]. For each $\bar{\boldsymbol{x}}_n$, the PQ linear scan simply retrieves the closest of the $K$ centers $\{\bar{\boldsymbol{\mu}}_k\}_{k=1}^K$ linearly using Eq. (3). This computation requires a cost of $O(KM)$ for each $\bar{\boldsymbol{x}}_n$. This is sufficiently fast for a small $K$ value, but is not efficient if $K$ is large. The PQTable is a hash-table-based acceleration data structure, which is tailored for the efficient computation of $d_{SD}$. When the number of items ($K$) is small, the computational cost of managing and hashing the PQTable is larger than for the PQ linear scan. However, for large $K$ values, the PQTable is between $10^2$ and $10^5$ times faster than the PQ linear scan [23].

Given the input PQ codes $\bar{\mathcal{X}}$, it is not easy to decide which method to use, because the computational cost depends on $K$, $M$, and the distribution of vectors of the target dataset. We adopt a simple but effective approach. Given the PQ codes, we first evaluate both methods several times, and then select the faster of the two. We found that this simple selection method is also useful for Bk-means, and therefore we incorporated this technique into the Bk-means method for the evaluation.

### 4.2 Update step

Once each input PQ code is assigned to its nearest cluster center, we update each cluster center such that the sum of the errors within the cluster is minimized. For typical real-valued vectors, this can be achieved by computing the mean vector among all of the vectors in each cluster. However, no method is known for computing a "mean PQ code" from a set of PQ codes. Here, we define the mean PQ code as that which minimizes the sum of the symmetric distances to each PQ code within a cluster.

We can propose a naïve straightforward method. The naïve method is a brute-force approach, which is therefore slow. The experimental results show that this naïve method is sometimes even slower than the assignment step, as we will discuss in Sec. 5.2. Consequently, we develop an alternative method, called *sparse voting*. By reorganizing the items in the cluster, sparse voting achieves the same result as the naïve method, but more efficiently. This simple modification accelerates the computation significantly (10× to 50×).

**Naïve method:** Let us focus on the $k$th cluster. For simplicity, we refer to the PQ codes assigned to the cluster as $\{\bar{\boldsymbol{x}}_n\}_{n=1}^{N_k}$,

where $N_k \sim N/K$. The purpose here is to compute a new center $\bar{\boldsymbol{\mu}}_k = [\bar{\mu}_k^1, \ldots, \bar{\mu}_k^M]^\top \in \{1, \ldots, L\}^M$. Because each subspace is independent, we consider the $m$th subspace. Therefore, the problem is defined as follows: Given $N_k$ integers $\{\bar{x}_n^m\}_{n=1}^{N_k}$, where each $\bar{x}_n^m \in \{1, \ldots, L\}$, we calculate the "mean" code $\bar{\mu}_k^m \in \{1, \ldots, L\}$.

The straightforward brute-force approach tests all possible candidates. Subsequently, the best candidate that minimizes the sum of the errors within the cluster is determined as follows:

$$\bar{\mu}_k^m \leftarrow \underset{l \in \{1, \ldots, L\}}{\arg\min} \sum_{n=1}^{N_k} d_{SD}^m(\bar{x}_n^m, l)^2. \tag{6}$$

Using Eq. (3), we find that $d_{SD}^m(\bar{x}_n^m, l)^2 = A^m_{\bar{x}_n^m, l}$. Therefore, this can be computed by simply looking up the table. This naïve computation requires a cost of $O(LN_k)$.

**Sparse voting:** Next, we develop a fast alternative method, called sparse voting. By creating a histogram, we can efficiently compute Eq. (6). Given $\{\bar{x}_n^m\}_{n=1}^{N_k}$, we scan these, and create an $L$-dimensional histogram of frequency:

$$\boldsymbol{h} = [h_1, \ldots, h_L]^\top \in \mathbb{N}^L, \tag{7}$$

where $h_l$ denotes the frequency with which an integer $l$ appears in $\{\bar{x}_n^m\}_{n=1}^{N_k}$. This scanning process requires a cost of $O(N_k)$.

Using $\boldsymbol{h}$, Eq. (6) can be rewritten as

$$\bar{\mu}_k^m \leftarrow \underset{l \in \{1, \ldots, L\}}{\arg\min} \; v_l, \text{ where } [v_1, \ldots, v_L]^\top = A^m \boldsymbol{h}. \tag{8}$$

It is easy to show that the right-hand side of Eq. (6) is equivalent to that of Eq. (8) once they are expanded. The computational cost of Eq. (8) is $O(L^2)$.

Furthermore, if $\boldsymbol{h}$ is sparse, then the cost becomes $O(L\|\boldsymbol{h}\|_0)$, where $\|\boldsymbol{h}\|_0 \in \{0, \ldots, L\}$ denotes the number of nonzero elements in $\boldsymbol{h}$. Thus, the entire cost of sparse voting is $O(N_k + L\|\boldsymbol{h}\|_0)$. Although sparse voting is a simple trick, it accelerates the computation significantly.

**Analysis:** With both the naïve method and the sparse voting method, the final center $\bar{\boldsymbol{\mu}}_k = [\bar{\mu}_k^1, \ldots, \bar{\mu}_k^M]^\top$ is created by computing $\bar{\mu}_k^m$ for all $m$. Therefore, for each cluster, the computational costs of the naïve method and the sparse-voting method are $O(LMN_k)$ and $O(M(N_k + L\|\boldsymbol{h}\|_0))$, respectively. Finally, by summing $K$ clusters, we find that the total costs are $O(LMN)$ and $O(M(N + KL\|\boldsymbol{h}\|_0))$, respectively.

If the constant factor is the same, then sparse voting is faster when $N/K > \frac{L}{L-1}\|\boldsymbol{h}\|_0 \sim \|\boldsymbol{h}\|_0$. Because the PQ codes in the same cluster tend to be similar (owing to the nature of clustering), the histogram $\boldsymbol{h}$ tends to be sparse, and this condition is satisfied in many cases, as we discuss in Sec. 5.2.

## 4.3 Pseudocode

Algorithm 1 presents the pseudocode for PQk-means. The pipeline is extremely simple. The `Init()` function initializes the centers, by simply randomly picking up $K$ codes from the input codes. The `Check()` function decides the manner in which the nearest neighbors are found, whether by a PQ linear scan or with a PQTable. This can be achieved by simply running both methods 10 times with randomly sampled vectors. `BuildTable()` creates a PQTable. `FindNN()` and `UpdateCenter()` are explained in Sec. 4.1 and Sec. 4.2, respectively. The results of the assignment function $a(n)$ are stored in an array. Any condition can be adopted as a stop condition.

**Algorithm 1:** PQk-means clustering

**Input:** $\bar{\mathcal{X}} = \{\bar{\boldsymbol{x}}_n\}_{n=1}^N$, // PQ codes
$\mathcal{A} = \{A^m\}_{m=1}^M$, // Distance matrices
$K$. // The number of clusters
**Output:** $\bar{\mathcal{M}} = \{\bar{\boldsymbol{\mu}}_k\}_{k=1}^K$. // PQ codes

```
M̄ ← Init(X̄)
flag ← Check(X̄, A)
repeat
    a ← ∅      // Array
    if flag then
        table ← BuildTable(M̄, A)   // PQTable
    for n ← 1 to N do
        if flag then
            // Sec. 4.1 (PQTable)
            a[n] ← FindNN(x̄_n, table)
        else
            // Sec. 4.1 (PQ linear scan)
            a[n] ← FindNN(x̄_n, M̄, A)
    for k ← 1 to K do
        μ̄_k ←UpdateCenter(X̄, a, A)   // Sec. 4.2
until stop condition;
```

## 5 EXPERIMENTAL RESULTS

We evaluated PQk-means using various datasets. All experiments were performed on a server with 3.0 GHz Intel Xeon CPUs (4 cores, 8 threads) and 128 GB of RAM [3]. For a fair comparison with existing methods, we employed a single-thread implementation for clustering (Sec. 5.2, 5.3, 5.4, and 5.5). For large-scale clustering (Sec. 5.6), we used a multithread implementation, in order to highlight the best performance. All source codes are publicly available on `https://github.com/DwangoMediaVillage/pqkmeans`.

### 5.1 Setup

**Compared methods:** For comparison, we implemented standard k-means clustering [22], Bk-means [14] with iterative quantization (ITQ) [13], and Ak-means [31] using FLANN [24]. Ak-means is an accelerated version of k-means, where the assignment step is accelerated using a KD tree. In addition, we compared our method with IQ-means [2], which is the latest subset-based method. Note that k-means, Ak-means, and IQ-means are **not** memory efficient for high-dimensional vectors.

**Datasets and features:** We used four datasets: ILSVRC2012, BIGANN, YFCC, and Deep1B. The details of each dataset are summarized in Table 2, where #test denotes the number of input vectors on which the clustering algorithms were applied. Likewise, #train denotes the number of vectors used for training the codewords for product quantization and the rotation matrices for ITQ.

[3] We verified that the largest experiment (Sec. 5.6) was also run on a computer with only 32 GB of RAM.

**Table 2: Dataset statistics.**

| Dataset | $D$ | #train | #test |
|---|---|---|---|
| ILSVRC_100C | 4,096 | 100K | 129,395 |
| ILSVRC_1000C | 4,096 | 100K | 1,281,167 |
| SIFT1M | 128 | 100K | 1,000,000 |
| SIFT1B | 128 | 1M | 1,000,000,000 |
| YFCC100M | 4,096 | 2M | 96,419,740 |
| Deep1B | 96 | 1M | 1,000,000,000 |

The ILSVRC2012 dataset is a subset of ImageNet [33]. This dataset consists of 1000 object categories, each of which contains around 1000 images. According to Gong et al. [14], the full dataset was named ILSVRC_1000C, and a small subset named ILSVRC_100C which was constructed by randomly picking 100 classes. For each image, we extracted a 4096D AlexNet feature [21], which was activated from the last hidden layer, using the chainer framework [37] with a pretrained model. We used 100K test images for training[4].

From BIGANN [19], we used the two datasets SIFT1M and SIFT1B. For the training of SIFT1B, we used the top one million vectors from the whole training set.

Yahoo's Flickr Creative Commons 100M (YFCC100M) dataset [36] contains around 100M images. An AlexNet feature vector was extracted from each image, as with ILSVRC. Two million randomly chosen features were used for training.

The Deep1B dataset [5] contains one billion test and 350M training features. Each feature was extracted from the last fully connected layer of GoogLeNet [35] for one billion images. The features were compressed to 96 dimensions using PCA, and $l_2$ normalized. For training, we used the top 1M vectors from the training set.

We used ILSVRC_100C, ILSVRC_1000C, and SIFT1M to compare the methods. Note that each dataset has a distinct nature. The AlexNet features have a larger dimension and a sparse nature, whereas the SIFT features are dense and structured. The datasets YFCC100M, SIFT1B, and Deep1B were used for the large-scale evaluation. Because the YFCC100M dataset includes the original images, the results of image clustering are evaluated visually (this will be illustrated later in Fig. 5).

**Encoding:** For feature encoding, we employed PQ [18] for PQk-means, and ITQ [13] for Bk-means [14]. The PQ codewords and ITQ rotation matrices were trained beforehand, using the training datasets. Subsequently, all of the features were converted to $B$-bit PQ codes for PQ, and $B$-bit binary strings for ITQ, where $B = 32$, 64, and 128. Note that $B = M \log_2 L = 8M$ for the PQ codes. Hereafter, we employ abbreviations to denote encodings with various bit lengths, e.g., "pqkmeans32" refers to 32-bit PQ encoding. Note that the bit length is a parameter specified by the user. Larger bit lengths improve the accuracy, but require more memory.

For each vector, the encoding of ITQ requires a cost of $O(D^2)$, and that of PQ requires $O(DL)$. The actual runtime of the encoding using ILSVRC_1000C was 522 s for ITQ, and 109 s for PQ. As discussed in Sec. 1, our assumption is that users only store encoded codes. Therefore, encoding is the preprocessing step used in this study. Note that IQ-means also requires a similar encoding process.

**Seed:** For the initial seeds of the clustering, we randomly sampled $K$ vectors from the input dataset. We fixed seeds for all methods using the same conditions (dataset, $B$, and $K$), to ensure a fair comparison. Note that in our preliminary study, we observed that the selection of seeds did not significantly affect the results [5].

## 5.2 Runtime analysis

We evaluated the runtime of the proposed PQk-means clustering method. Table 3 presents a runtime comparison for each step in the assignment, the update using the naïve method, and the update using the proposed sparse-voting scheme. The results confirm the following points. First, **the proposed sparse voting method is always faster than the naïve updating method, with a large margin** (e.g., 54× faster in ILSVRC_1000C with $K = 10^3$). Second, **the naïve updating method is sometimes even slower than the assignment step** (for ILSVRC_100C with $K = 10^2$, ILSVRC_1000C with $K = 10^3$, and SIFT1M with $K = 10^2$).

These results indicate that the proposed sparse-voting scheme is highly efficient, even though it achieves the same accuracy as the naïve method. Consequently, we employed sparse voting during the subsequent evaluations in this study.

Theoretically, the runtime of PQk-means is

$$\min(O(KMN), O(NT_{table})) + O(M(N + KL\|\boldsymbol{h}\|_0)). \quad (9)$$

The first term corresponds to the assignment step, whether using a PQ linear scan ($O(KMN)$) or a PQTable ($O(NT_{table})$). Note that $T_{table}$ denotes the cost of a search for nearest neighbors using the PQTable, which is heavily dependent on the data distribution. The second term corresponds to the updating step using the sparse-voting scheme. Because of the efficiency of updating cluster centers using the sparse voting scheme, as shown in Table 3, the dominant step is the assignment.

Note that the runtime of PQk-means does not depend on the dimension $D$ of the original vectors, meaning that our PQk-means method performs efficiently for high-dimensional vectors.

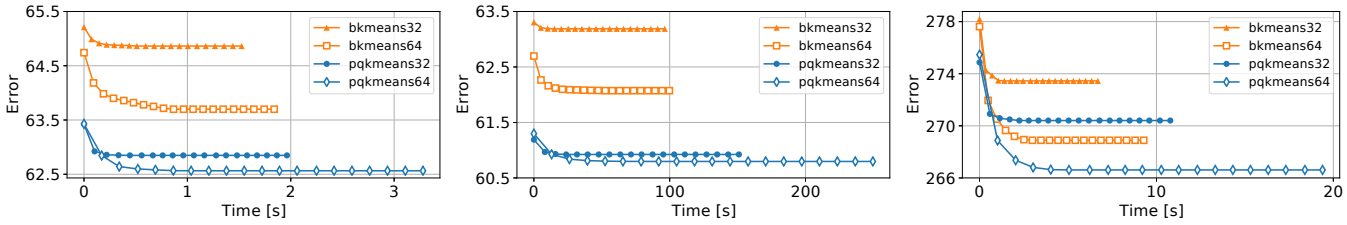
## 5.3 Memory consumption

For $B$-bit codes, PQk-means requires $\frac{B}{8}(N + K)$ bytes for codes and centers, and $4L^2M$ bytes for distance matrices. In addition, PQk-means requires an array of lists to specify an assignment ($a$ in Algorithm 1), which requires $4N$ bytes in total. If the PQTable is used, this requires $4K \cdot 2^{Q(\log_2(B/\log_2 K))}$ bytes [23], where $Q()$ is a rounding operation. The sum mentioned above constitutes the theoretical runtime memory consumption. Usually, $N$ is significantly larger than $K$. Therefore, **the main contributors to the memory are the input codes and an assignment array,** $(B/8 + 4)N$.

Compared to the standard k-means and the Ak-means methods, both of which require at least $4D(N+K)$ bytes for codes and centers, the proposed PQk-means requires significantly less memory. Because the memory consumption of the PQk-means does not depend on the dimension $D$ of the original vectors, PQk-means is particularly memory efficient when $D$ is large, such as for AlexNet features ($D = 4096$). For example, input vectors from ILSVRC_1000C require

[4]Because ILSVRC2012 is used for image-recognition competitions, it contains more training images than test images. However, as our objective is clustering, we reversed the two groups, using the training images as test images and vice versa.

[5]For example, ten trials showed that the mean error is 242 and the standard deviation is 0.12 for SIFT1M with $K = 10^3$ using 32-bit codes.




(a) ILSVRC_100C with $K = 10^2$ (b) ILSVRC_1000C with $K = 10^3$ (c) SIFT1M with $K = 50$

**Figure 2: Comparison of PQk-means with Bk-means in terms of errors and runtime. Errors are plotted for each iteration. All lines are plotted for 20 iterations. A relatively longer line indicates that more time was required.**

**Table 3: Runtime comparison for each step using 32-bit codes for various conditions. For each condition, we run PQk-means twice (w/ Naïve or w/ sparse voting), and report the runtimes. For each condition, the most/least time-consuming step is highlighted using a bold/underlined font. We also report a macro average value of $\|\boldsymbol{h}\|_0$. All values constitute averages over 20 iterations.**

| | | | | Update [ms] | |
|---|---|---|---|---|---|
| Dataset | $K$ | $\|\boldsymbol{h}\|_0$ | Assignment [ms] | Naïve | Sparse |
| ILSVRC_100C | $10^2$ | 64.0 | 91.1 | **301** | 5.05 |
| | $10^3$ | 18.7 | **795** | 183 | 10.1 |
| ILSVRC_1000C | $10^2$ | 133 | $1.09 \times 10^3$ | $\mathbf{4.72 \times 10^3}$ | 124 |
| | $10^3$ | 52.7 | $\mathbf{8.51 \times 10^3}$ | $5.29 \times 10^3$ | 98.2 |
| | $10^4$ | 14.0 | $\mathbf{3.86 \times 10^4}$ | $2.03 \times 10^3$ | 218 |
| SIFT1M | $10^2$ | 145 | 811 | $\mathbf{4.26 \times 10^3}$ | 105 |
| | $10^3$ | 77.1 | $\mathbf{6.21 \times 10^3}$ | $2.43 \times 10^3$ | 103 |

$1,281,167 \times 4096 \times 4 = 21$ GB. By contrast, PQ codes with 32 bits require only $1,281,167 \times 32/8 = 5.12$ MB. This confirms the advantages of using short-code encoding schemes. As shown in Sec. 5.4, even when features are encoded as very short codes, the clustering performance declines slightly, with a substantial speed-up. Notably, Bk-means offers a comparable advantage in terms of memory.

## 5.4 Detailed comparison with Bk-means

We compared the proposed PQk-means method with Bk-means, which is the closest comparable method (see Table 1). We examined the behavior of both methods at each iteration, especially for relatively small $K$ values. Because $K$ is small, the linear scan was used in the assignment step for both methods. The results highlighted the general tendencies that PQk-means is more accurate, whereas Bk-means is faster.

Clustering errors were computed as follows. Let us assume that either PQk-means or Bk-means is applied to the short codes to create $K$ clusters. Following clustering, the corresponding **original** vectors $\{\boldsymbol{x}_n\}_{n=1}^N$ are collected. Subsequently, the error $E$ for the original vectors is computed using Eq. (1). As $E$ measures the average errors in the original vectors (rather than codes), we can compare the results of PQk-means using those of Bk-means.

Figure. 2 presents the runtimes and errors during each iteration. We obtained some interesting results.

**PQk-means vs. Bk-means**: In the comparison of PQk-means with Bk-means for the same code length, the former always achieved smaller errors. This is because the employed product quantization is more accurate than ITQ, as reported in [16]. In terms of the runtime, Bk-means was always faster than PQk-means for the same code length. This is because comparing bit strings is faster than comparing two PQ codes, which also constitutes expected behavior [16].

**Code length**: When considering different code lengths, there were smaller errors for longer bit lengths, as expected. Interestingly, **the results for pqkmeans32 were more accurate than those of bkmeans64** in Fig. 2a and Fig. 2b. This could be explained by the higher expressiveness of PQ compared with that of ITQ.

**Convergence behavior**: As can be observed, 20 iterations were sufficient to achieve convergence in all cases. Note that if we stop the iteration when the error does not change from the previous iteration, PQk-means can achieve similar computational cost as Bk-means for these datasets.

## 5.5 Comparison with existing methods under several conditions

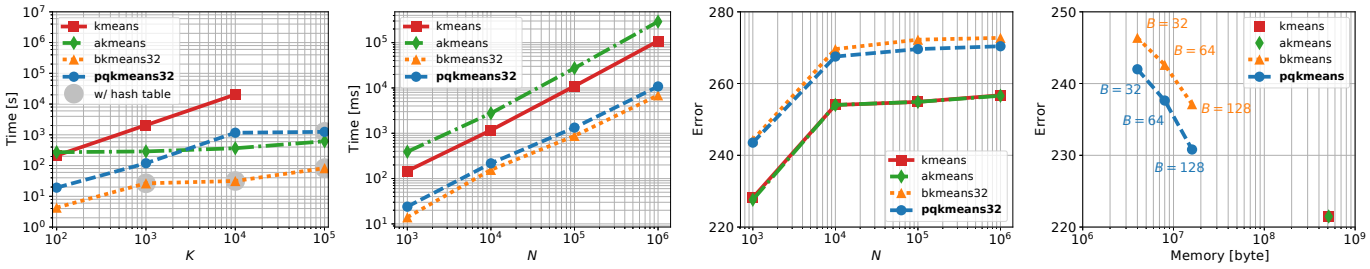
We compared the proposed PQk-means with Bk-means, k-means, and Ak-means under several conditions. Our findings are summarized as follows:

- k-means was between 10× and 1,000× slower than PQk-means.
- Ak-means was accurate. However, it was slow for large $D$ and/or relatively small $K$ values.
- k-means and Ak-means required between 100× and 4,000× more memory than PQk-means.
- Short codes, such as 32-bit PQ codes, were effective in terms of the balance of accuracy, memory cost, and runtime.

**SIFT1M:** Figure. 3 illustrates the relationship between the runtime, errors, and memory consumption according to $N$ or $K$ using SIFT1M. As expected, k-means clustering resulted in the fewest errors in all cases (Fig. 3c). However, it was more than ten times slower in all cases compared with PQk-means and Bk-means (Fig. 3a).

Figure. 3c shows that Ak-means achieved low errors (almost the same as k-means). However, owing to the overhead of the approximated search, Ak-means was slow for relatively small $K$ (Fig. 3a), with Ak-means being slowest method for $K = 50$ (Fig. 3b).

Figure. 3d shows that Ak-means and k-means were not memory efficient, even though these methods achieved lower errors. Ak-means and k-means consumed 512 MB memory space for the vectors, whereas PQk-means and Ak-means required only $B/8$ MB.




(a) Runtime according to $K$. (b) Runtime according to $N$ ($K = 50$). (c) Error according to $N$ ($K = 50$). (d) Error according to memory ($K = 10^3$).

**Figure 3: Relation between the errors, runtime, and memory consumption of the input vectors, according to $N$ or $K$, for the SIFT1M dataset after 20 iterations. The gray dots in (a) indicate that hash tables ([29] for Bk-means and [23] for PQk-means) were used in the assignment step.**

Fig. 3d also shows that the error of PQk-means for 32-bit codes was lower than that of Bk-means for 64-bit codes.

**ILSVRC_1000C:** A comparison using ILSVRC_1000C with 32-bit codes is summarized in Table 4. To compare the results more intuitively, we introduce an additional evaluation criterion, the Rand index [32]. Given a pair of clustering results, the Rand index computes the similarity between them. We compared the result of k-means against each method, where a higher Rand index indicates a higher similarity.

PQk-means was superior to Bk-means in terms the Rand index with the same code length. For example, the Rand index of PQk-means (0.142) was higher than that of Bk-means (0.046) for $K = 10^3$.

The errors of Ak-means were close to those of k-means. This was also confirmed by the high Rand index (e.g., 0.465 for $K = 10^2$). However, Ak-means required a huge amount of memory (21.0 GB, whereas 5.12 MB was required for PQk-means and Bk-means). Moreover, because the runtime of Ak-means depends on the dimension of the vectors, Ak-means was slower for high-dimensional features, such as AlexNet. Table 4 shows that Ak-means was between 5× and 164× slower than PQk-means for all $K$.

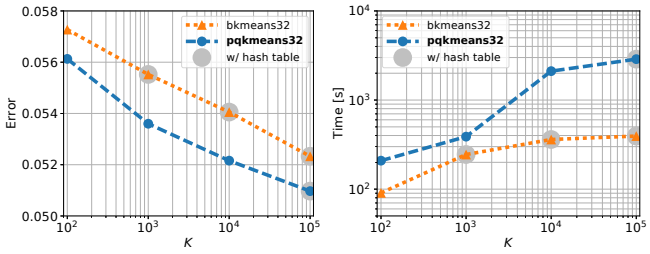
Interestingly, although the PQk-means with 32-bit codes was 20 times faster and required 4,000 times less memory than Ak-means, the Rand index of PQk-means (0.142) is slightly lower than that of Ak-means (0.2) for $K = 10^3$. This implies that for the purpose of clustering, the short-length code (e.g., 32-bit) can provide a strong balance between the accuracy, memory consumption, and runtime. We believe that this is a practically important result for clustering in memory-restricted environments.

## 5.6 Large-scale clustering evaluation

In this section, we present the results of a large-scale evaluation using three billion-scale datasets, namely the YFCC100M, SIFT1B, and Deep1B datasets. For the three datasets, we ran PQk-means with 32-bit codes and various values of $K$ using a parallel implementation on a single machine. In addition, we ran Bk-means with a parallel implementation for YFCC100M. To highlight the best performance, we stopped the iteration when the error converged. The number of iterations required for convergence was five for all datasets. Because these datasets are extremely large (1.58 TB, 512 GB, and 384 GB, for YFCC100M, SIFT1B, and Deep1B, respectively), an ordinary computer cannot store all of the original data in its memory simultaneously.

**Table 4: Comparison of methods using the ILSVRC_1000C dataset with 32-bit codes after 20 iterations.**

| Method | $K$ | Error | Rand index | Time [s] | Memory |
|---|---|---|---|---|---|
| PQk-means | $10^2$ | 65.09 | 0.230 | 18.2 | 5.12 MB |
| | $10^3$ | 60.92 | 0.142 | $1.51 \times 10^2$ | |
| | $10^4$ | 59.03 | - | $7.22 \times 10^2$ | |
| Bk-means | $10^2$ | 66.35 | 0.111 | 12.3 | 5.12 MB |
| | $10^3$ | 63.19 | 0.046 | $1.00 \times 10^2$ | |
| | $10^4$ | 60.70 | - | $1.15 \times 10^2$ | |
| k-means | $10^2$ | 64.25 | 1.0 | $9.12 \times 10^3$ | 21.0 GB |
| | $10^3$ | 58.95 | 1.0 | $1.06 \times 10^5$ | |
| Ak-means | $10^2$ | 64.29 | 0.465 | $3.00 \times 10^3$ | 21.0 GB |
| | $10^3$ | 59.76 | 0.200 | $2.96 \times 10^3$ | |
| | $10^4$ | 56.78 | - | $3.65 \times 10^3$ | |

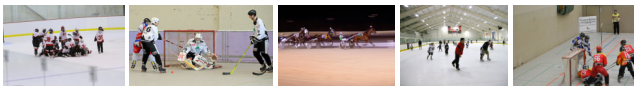



(a) Error according to $K$ (b) Time according to $K$

**Figure 4: Large-scale clustering evaluation for the YFCC100M dataset with five iterations.**

**YFCC100M:** Fig. 4 presents a comparison between PQk-means and Bk-means. As discussed in Sec. 5.4 and Sec. 5.5, PQk-means always achieved more accurate clustering and Bk-means was always faster for the same $K$.

The resulting images for the clustering with PQk-means with $B = 32$ and $K = 10^5$ are presents in Fig. 5. These results show

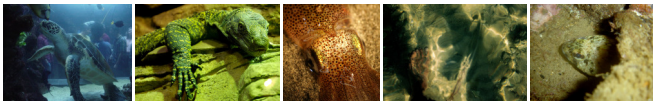


Cluster id: 5703 (a sports game on ice)

Cluster id: 95307 (a European-style church)


Cluster id: 17713 (some texts)

Cluster id: 9566 (a palm tree)



Cluster id: 76803 (a creature of the sea)

**Figure 5: Example images from image clustering using PQk-means with $B = 32$ and $K = 10^5$ for the YFCC100M dataset. Each row shows images belonging to the same cluster.**

that PQk-means successfully clustered the images. The images in each cluster show a consistent scenario, as follows: cluster ID 5703 shows a sports game on ice, cluster ID 95307 a European-style church, cluster ID 17713 some texts, cluster ID 9566 a palm tree, and cluster ID 76803 a sea creature. From these results, we conclude that clustering using only 32-bit codes can provide useful results.

**SIFT1B and Deep1B:** Table 5 presents the runtime evaluation for SIFT1B and Deep1B. Remarkably, the runtime results for SIFT1B and Deep1B exhibit similar behavior, even though the data distribution of SIFT features and GoogLeNet features would be different. These results indicate that we can predict the runtime performance of PQk-means. This is important, because estimating the runtime of large-scale clustering is usually difficult.

Although the required memory was less than 32 GB, PQk-means can handle $10^9$ vectors with $K = 10^5$ in around just half a day (14 hours for SIFT1B and 12 hours for Deep1B). This implies that PQk-means allows practical large-scale clustering on a single machine.

## 5.7 Discussions

**Comparison with Bk-means:** The comparative studies illustrated that both PQk-means and Bk-means are less accurate than the original k-means method. However, they are both considerably faster, and use significantly less memory.

PQk-means was more accurate than Bk-means in all settings. Remarkably, PQk-means with 32-bit codes sometimes achieved a better accuracy than Bk-means with 64-bit codes (Figs. 2a, 2b, and 3d). In terms of the computational cost, Bk-means was faster than

**Table 5: Large-scale clustering evaluation of PQk-means for the SIFT1B and Deep1B datasets with five iterations ($B = 32$).**

| Dataset | $N$ | $K$ | Error | Time [s] | w/ table |
|---|---|---|---|---|---|
| SIFT1B | $10^9$ | $10^2$ | 303.4 | $1.88 \times 10^3$ (31 m) | |
| | | $10^3$ | 277.4 | $3.95 \times 10^3$ (66 m) | |
| | | $10^4$ | 256.0 | $3.68 \times 10^4$ (10 h) | |
| | | $10^5$ | 235.1 | $5.14 \times 10^4$ (14 h) | ✓ |
| Deep1B | $10^9$ | $10^2$ | 0.800 | $1.98 \times 10^3$ (33 m) | |
| | | $10^3$ | 0.741 | $4.04 \times 10^3$ (67 m) | |
| | | $10^4$ | 0.697 | $3.68 \times 10^4$ (10 h) | |
| | | $10^5$ | 0.655 | $4.47 \times 10^4$ (12 h) | ✓ |

PQk-means, especially for large $K$. This difference stems from the fast search using hash tables [29], which was faster than using the PQTable [23] for PQ codes. The next step should be to improve this assignment step using an even more efficient data structure.

An important advantage of PQ codes is that the original vectors can be approximately reconstructed from the PQ codes.

**Comparison with Ak-means:** Compared with PQk-means for the same value of $K$, Ak-means achieved lower errors. However, it was slower, especially for relatively small values of $K$ (Fig. 3a, Fig. 3b) or large values of $D$ (Table 4). Because Ak-means stores the original $D$-dimensional vectors, it requires significantly more memory space than PQk-means. The advantage of Ak-means is that it does not require an encoding step. Ak-means would be useful for relatively small-scale problems, where all of the original vectors can be stored in the memory.

**Comparison with IQ-means:** IQ-means is an accelerated version of ranked-retrieval [8] that skips distance computations when vectors are placed far away from centers. IQ-means can be the fastest clustering method for large-scale data. However, IQ-means seems not memory-efficient, and its accuracy was much lower than PQk-means for the YFCC100M dataset. Please refer to our supplementary material for the discussion on IQ-means.

## 6 CONCLUSIONS

In this paper, we introduced the PQk-means clustering method, which is a billion-scale clustering algorithm for PQ codes. The proposed method consists of two steps: an assignment step using a PQTable and an update step with a sparse-voting scheme. PQk-means can cluster even high-dimensional vectors efficiently, because the runtime and memory cost do not depend on the dimensions of the original vectors. For the same code length, the accuracy of PQk-means was shown to be consistently superior to that of Bk-means, with additional an computational cost. Experimental results demonstrated that the PQk-means achieved billion-scale clustering within around half a day.

The next step will be to boost PQk-means by using GPUs. Among the widespread applications of GPU, GPU-based acceleration is becoming a promising method for large-scale clustering [20]. Because PQk-means is simple and easy to parallelize, its performance can be boosted using GPUs.

**Acknowledgments:** This work was supported by JST ACT-I Grant Number JPMJPR16UO, Japan.

## REFERENCES


[1] David Arthur and Sergei Vassilvitskii. 2007. k-means++: The Advantages of Careful Seeding. In *Proc. ACM SODA*.

[2] Yannis Avrithis, Yannis Kalantidis, Evangelos Anagnostopoulos, and Ioannis Z. Emiris. 2015. Web-scale Image Clustering Revisited. In *Proc. IEEE ICCV*.

[3] Artem Babenko and Victor Lempitsky. 2014. Additive Quantization for Extreme Vector Compression. In *Proc. IEEE CVPR*.

[4] Artem Babenko and Victor Lempitsky. 2015. Tree Quantization for Large-Scale Similarity Search and Classification. In *Proc. IEEE CVPR*.

[5] Artem Babenko and Victor Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *Proc. IEEE CVPR*.

[6] Olivier Bachem, Mario Lucic, Hamed Hassani, and Andreas Krause. 2016. Fast and Provably Good Seedings for k-Means. In *Proc. NIPS*.

[7] Olivier Bachem, Mario Lucic, S. Hamed Hassani, and Andreas Krause. 2016. Approximated K-Means++ in Sublinear Time. In *Proc. AAAI*.

[8] Andrei Broder, Lluis Garcia-Pueyo, Vanja Josifovski, Sergei Vassilvitskii, and Srihari Venkatesan. 2014. Scalable K-Means by Ranked Retrieval. In *Proc. ACM WSDM*.

[9] Qi Dai, Jianguo Li, Jingdong Wang, and Yu-Gang Jiang. 2016. Binary Optimized Hashing. In *Proc. MM*.

[10] Matthijs Douze, Hervé Jégou, and Florent Perronnin. 2016. Polysemous Codes. In *Proc. ECCV*.

[11] Charles Elkan. 2003. Using the Triangle Inequality to Accelerate k-Means. In *Proc. ICML*.

[12] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE TPAMI* 36, 4 (2014), 744–755.

[13] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2013. Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-scale Image Retrieval. *IEEE TPAMI* 35, 12 (2013), 2916–2929.

[14] Yunchao Gong, Marcin Pawlowski, Fei Yang, Louis Brandy, Lubomir Bourdev, and Rob Fergus. 2015. Web Scale Photo Hash Clustering on A Single Machine. In *Proc. IEEE CVPR*.

[15] Robert M. Gray. 1984. Vector Quantization. *IEEE ASSP Magazine* 1, 2 (1984), 4–29.

[16] Kaiming He, Fang Wen, and Jian Sun. 2013. K-means Hashing: an Affinity-Preserving Quantization Method for Learning Binary Compact Codes. In *Proc. IEEE CVPR*.

[17] Anil K. Jain. 2010. Data Clustering: 50 Years Beyond K-means. *Pattern Recognition Letters* 31, 8 (2010), 651–666.

[18] Hervé Jégou, Matthijis Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE TPAMI* 33, 1 (2011), 117–128.

[19] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in One Billion Vectors: Re-rank with Souce Coding. In *Proc. IEEE ICASSP*.

[20] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale Similarity Search with GPUs. *CoRR* abs/1702.08734 (2017).

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet Classification with Deep Convolutional Neural Networks. In *Proc. NIPS*.

[22] Stuart P. Lloyd. 1982. Least Squares Quantization in PCM. *IEEE TIT* 28, 2 (1982), 129–137.

[23] Yusuke Matsui, Toshihiko Yamasaki, and Kiyoharu Aizawa. 2015. PQTable: Fast Exact Asymmetric Distance Neighbor Search for Product Quantization using Hash Tables. In *Proc. IEEE ICCV*.

[24] Marius Muja and David G. Lowe. 2014. Scalable Nearest Neighbour Algorithms for High Dimensional Data. *IEEE TPAMI* 36, 11 (2014), 2227–2240.

[25] James Newling and François Fleuret. 2016. Fast K-Means with Accurate Bounds. In *Proc. ICML*.

[26] James Newling and François Fleuret. 2016. Nested Mini-Batch K-Means. In *Proc. NIPS*.

[27] David Nistér and Henrik Stewénius. 2006. Scalable Recognition with a Vocabulary Tree. In *Proc. IEEE CVPR*.

[28] Mohammad Norouzi and David J. Fleet. 2013. Cartesian k-means. In *Proc. IEEE CVPR*.

[29] Mohammad Norouzi, Ali Punjani, and David J. Fleet. 2014. Fast Exact Search in Hamming Space With Multi-Index Hashing. *IEEE TPAMI* 36, 6 (2014), 1107–1119.

[30] Eng-Jon Ong and Miroslaw Bober. 2016. Improved Hamming Distance Search Using Variable Length Substrings. In *Proc. IEEE CVPR*.

[31] James Philbin, Ondřej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. 2007. Object Retrieval with Large Vocabularies and Fast Spatial Matching. In *Proc. IEEE CVPR*.

[32] William M. Rand. 1971. Objective Criteria for the Evaluation of Clustering Methods. *J. Amer. Statist. Assoc.* 66, 336 (1971), 846–850.

[33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* (2015), 1–42.

[34] David Sculley. 2010. Web-Scale K-Means Clustering. In *Proc. ACM WWW*.

[35] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going Deeper With Convolutions. In *Proc. IEEE CVPR*.

[36] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. 2016. YFCC100M: The New Data in Multimedia Research. *Commun. ACM* 59, 2 (2016), 64–73.

[37] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. 2015. Chainer: A Next-generation Open Source Framework for Deep Learning. In *Proc. Workshop on Machine Learning Systems*.

[38] Jing Wang, Jingdong Wang, Qifa Ke, Gang Zeng, and Shipeng Li. 2012. Fast Approximate k-Means via Cluster Closures. In *Proc. IEEE CVPR*.

[39] Jianfeng Wang, Jingdong Wang, Jingkuan Song, Xin-Shun Xu, Heng Tao Shen, and Shipeng Li. 2015. Optimized Cartesian K-Means. *IEEE TKDE* 27, 1 (2015), 180–192.

[40] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. 2017. A Survey on Learning to Hash. *IEEE TPAMI* PP, 99 (2017), 1–1.

[41] Jianwei Yang, Devi Parikh, and Dhruv Batra. 2016. Joint Unsupervised Learning of Deep Representations and Image Clusters. In *Proc. IEEE CVPR*.

[42] Ting Zhang, Chao Du, and Jingdong Wang. 2014. Composite Quantization for Approximate Nearest Neighbor Search. In *Proc. ICML*.

[43] Ting Zhang, Guo-Jun Qi, Jinhui Tang, and Jingdong Wang. 2015. Sparse Composite Quantization. In *Proc. IEEE CVPR*.