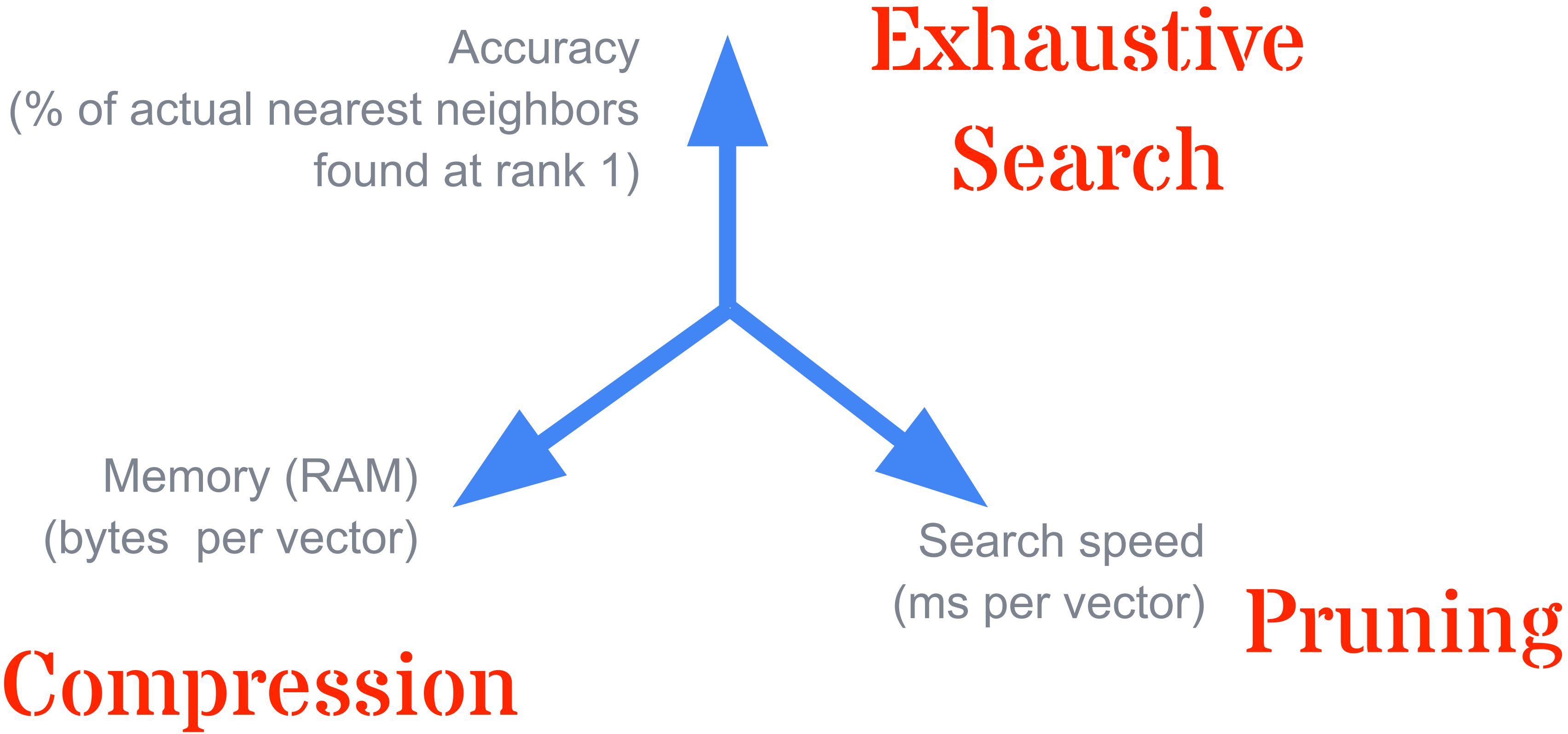
Vector search #8 – Quantization for lossy vector compression

Tradeoffs of vector search

3-handed tradeoff



In this class

- Mainly about the “compression” hand
- Fix size of representation
  - Because RAM is constrained
- Operating points between the two other hands

- Examples from Faiss
  - Implements many index types
  - Explore boundaries

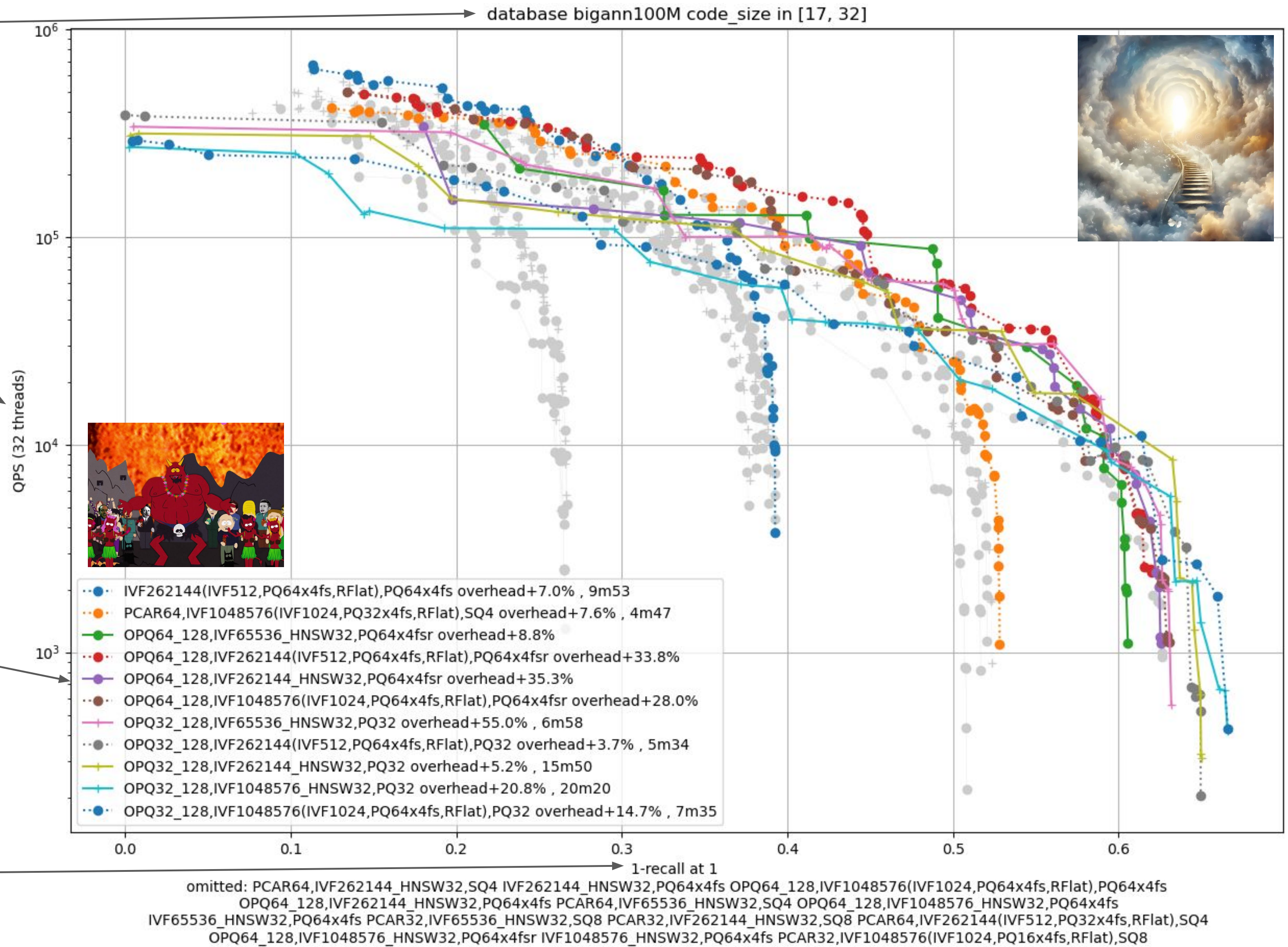
Next page: Faiss wiki
https://github.com/facebookresearch/faiss/wiki/Indexing-1G-vectors

Memory budget: max 32 bytes per vector

Speed: Queries per second (Log scale)

Each curve = one type of index

Accuracy: 1-recall@1



Built in conjunction with a notebook

- This sign:
- Means there is related content in the notebook.

Vector quantization

Vector quantization: definition

- Quantization: map a vector to an integer
  - Input is (supposed to be) continuous
  - Output is discrete
- Integer ≡ bit array of fixed size ≡ code
- Reconstruction: inverse map
  - We recover only an approximation
  - The reconstruction is lossy
- Evaluation: Mean Squared Error
  - Because it has nice arithmetic properties…
  - Invariant with d-dim rotation

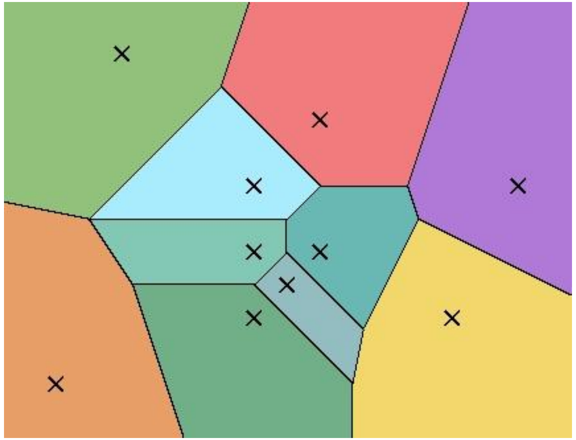
$$q : \mathbb{R}^d \mapsto \{0, ..., k-1\}$$

$$r : \{0, ..., k-1\} \mapsto \mathbb{R}^d$$

$$\mathrm{MSE} = \mathbb{E}_x \left[\| r(q(x)) - x \|^2 \right]$$

Codes and relationship with clustering

- Numbering is sequential
  - Otherwise do a mapping in the discrete domain
- Size of codes $\lceil \log_2(k) \rceil$



- Quantization cell
  - Locus of vectors that produce the same code

$$q^{-1}(\{i\}), \;\; i \in \{0, \ldots, k-1\}$$

- All quantization cells are a partition of input space
- From a discrete point of view: clustering
  - Reconstruction values are called “centroids”

Lloyd’s optimality conditions (reminder)

- To minimize the MSE for discrete sets
- 1: a vector should be assigned to the nearest reconstruction
    - Otherwise re-assign that vector: decrease MSE!
- 2: each centroid should be the center of mass of points assigned to it
    - Otherwise just move the center of mass: decrease MSE!
- Necessary, not sufficient

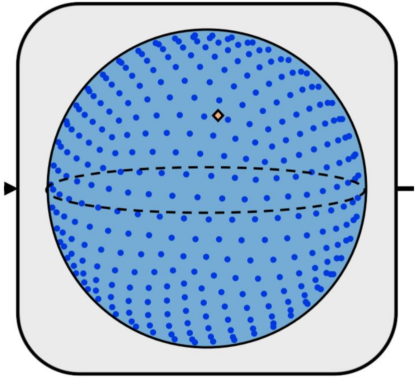
- The k-means algorithm
    - Iterate the two optimality conditions

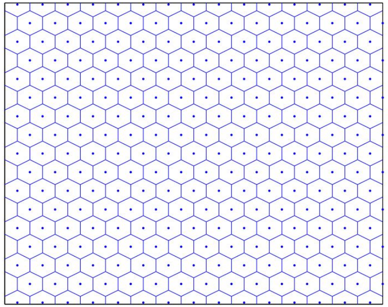
Cases where the optimal centroids are known

- Uniform distribution
    - A_x Lattice
- Uniform over a sphere
    - Integer lattice on sphere
- Unknown for Gaussian data

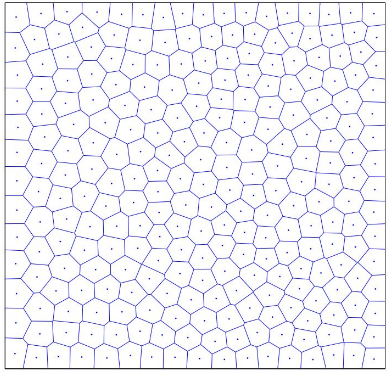
[Paulevé et al, Locality sensitive hashing: a comparison of hash function types and querying mechanisms, Pattern recog letters 2010]

[Sablayrolles et al, spreading vectors for similarity search, ICLR’19]

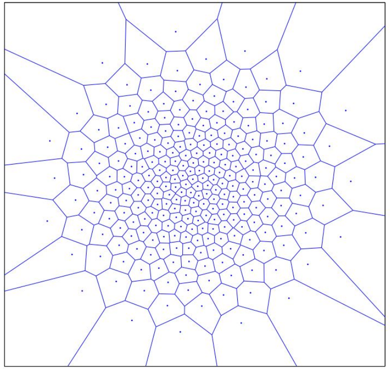




(b) A_2 lattice



(c) k-means
Uniform distribution



(d) k-means
Gaussian distribution

Figure 3: Voronoi regions associated with random projections (a), lattice A_2 (b) and a k-means quantizer (c,d).

How optimal is k-means?

Optimality of k-means: practical considerations

- On small scale, practical k-means is quite off from the global optimum
- On large scale we don't know!
  - NP-hard is very hard

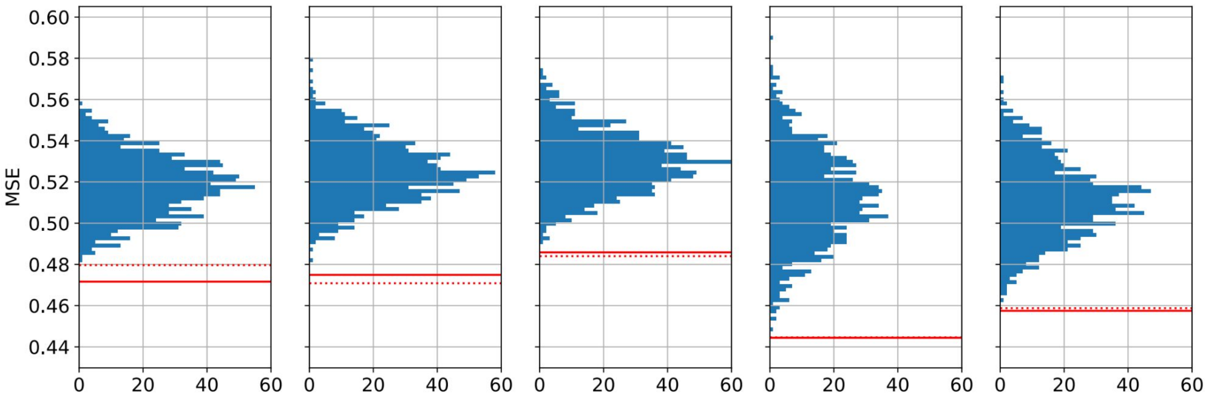
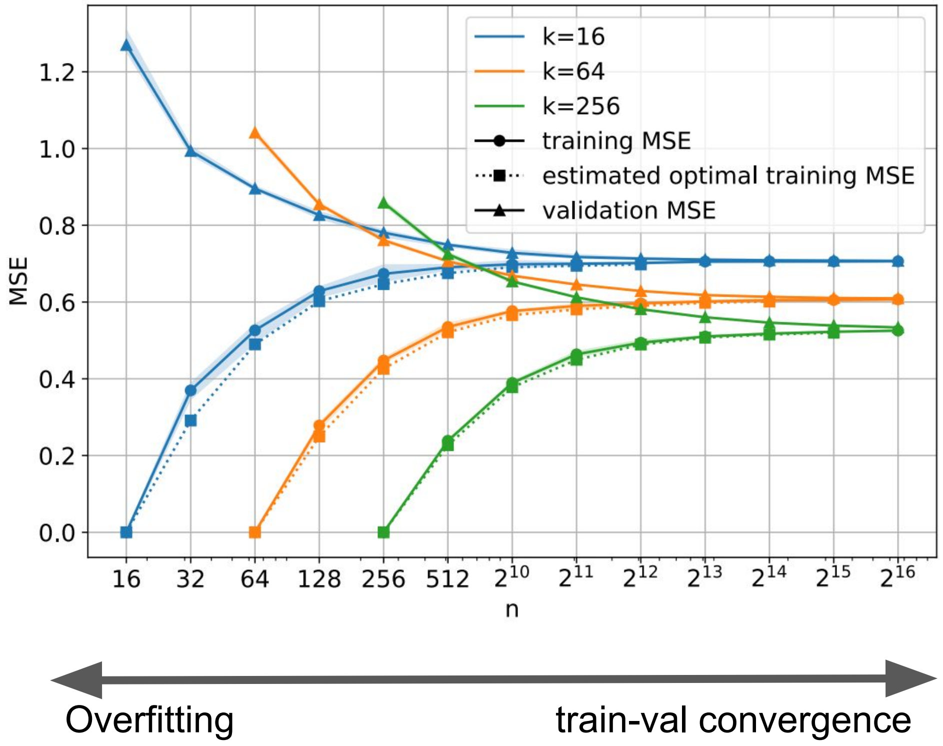


Figure 3: Histogram of MSE results for 1000 runs of k-means with $(n, k) = (24, 8)$. The experiment is run on 5 subsets of the same data distribution, hence the 5 plots. The exact global optimum is indicated in red. The estimate of this optimum from the k-means runs is the red dashed line.

[Augmented k-means, Touvron, Douze, Jégou, unpublished]

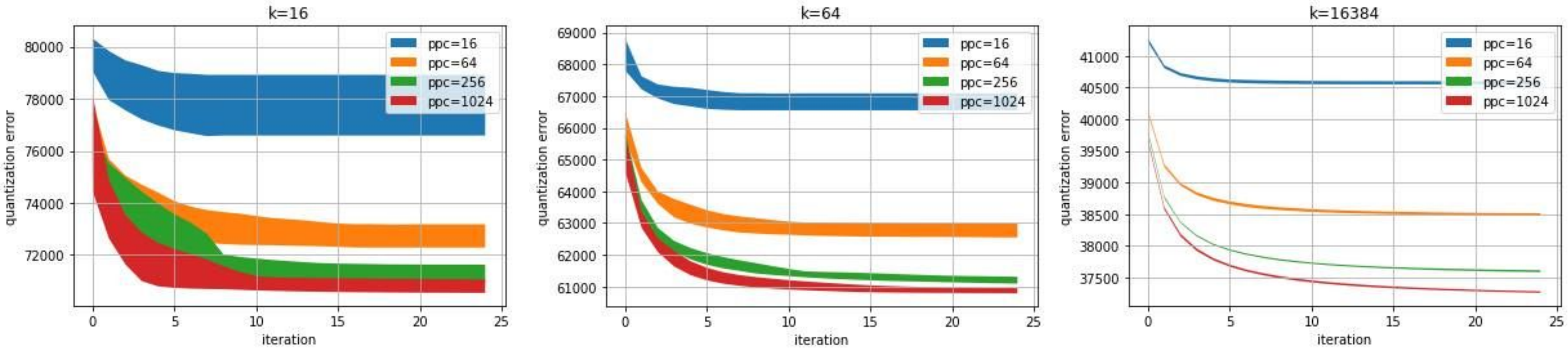
Difference between training and validation

- MSE can be computed on
  - Training vectors → Lloyd's conditions
  - Validation → how it's going to be used in reality
- Small and large-data regime
- Relevant parameter is n/k
  - Nb training points per centroid



Optimality of k-means: practical considerations

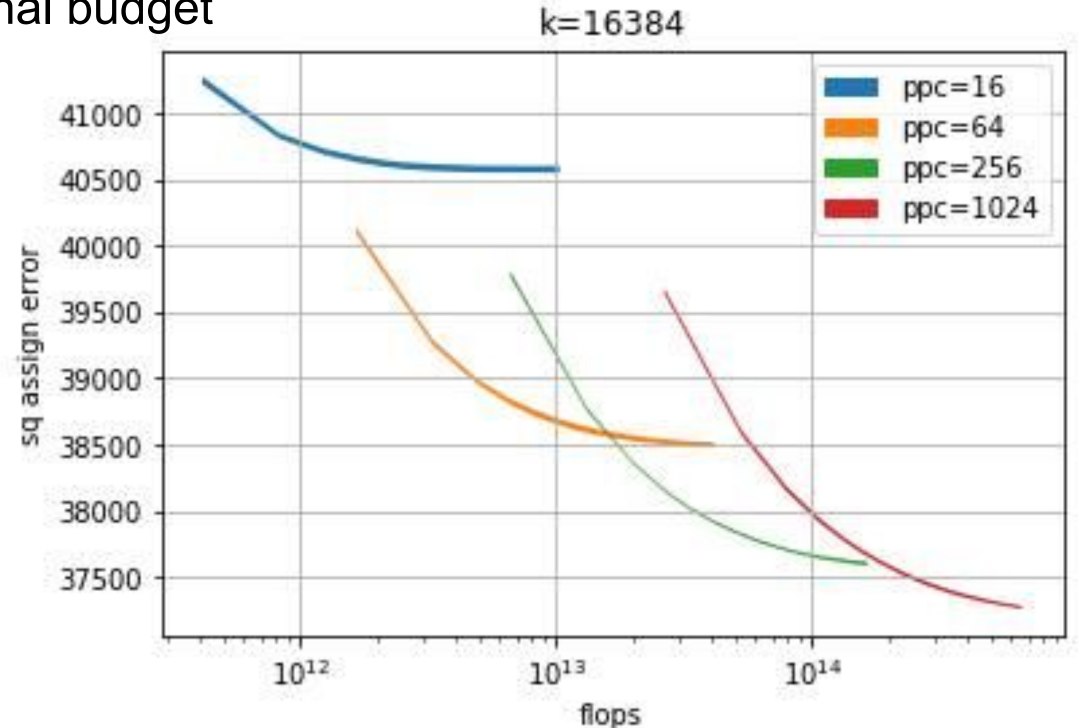
- As a function of the Points Per Centroid (ppc)
- Validation MSE
- Line thickness: min and max of 10 k-means runs



- It does not matter to have many initializations when k increases

Making good use of FLOPS

- Training a 100-centroid k-means on 1M vectors
    - Waste of resources?
- Given a certain computational budget
    - Balance nb of iterations and nb points per centroid



Scaling k-means

Scaling k-means

- Complexity
- Required to get larger codes
  - 3 bytes = 24 bits = 16M centroids
  - Not a very large code…
  - But a HUGE number of centroids
- Expensive stage is assignment
  - NNS problem
- Brute-force hardware scaling
  - CPU training
  - GPU training
  - Distributed training…

Matthijs Douze
March 8, 2019 ·

KMeans of 500M points to 10M centroids

144 dim, 20 iterations, 10x8 GPUs: 22h.

This is the largest k-means optimization we have done so far with Faiss. It required a version where the training set is distributed over 10 machines. The k-means logic was also re-implemented in Python. Nothing fancy, just brute force.

[Nistér Stewenius, Scalable recognition with a vocabulary tree, CVPR'06]
[Muja, Lowe, Scalable Nearest Neighbor Algorithms
for High Dimensional Data, PAMI'14]

Hierarchical k-means

- Inspired by bisection in 1D
- Run k-means recursively to subdivide
  - Iterate
- Often used for search
  - Basis of the FLANN library
- But sub-optimal in terms of MSE
  - See notebook

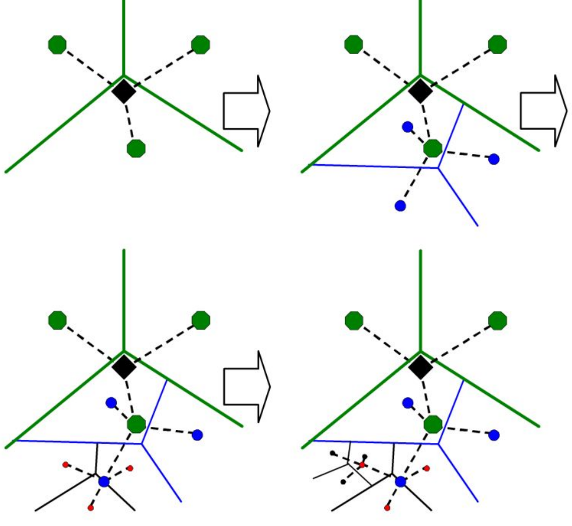


Figure 2. An illustration of the process of building the vocabulary tree. The hierarchical quantization is defined at each level by k centers (in this case $k = 3$) and their Voronoi regions.

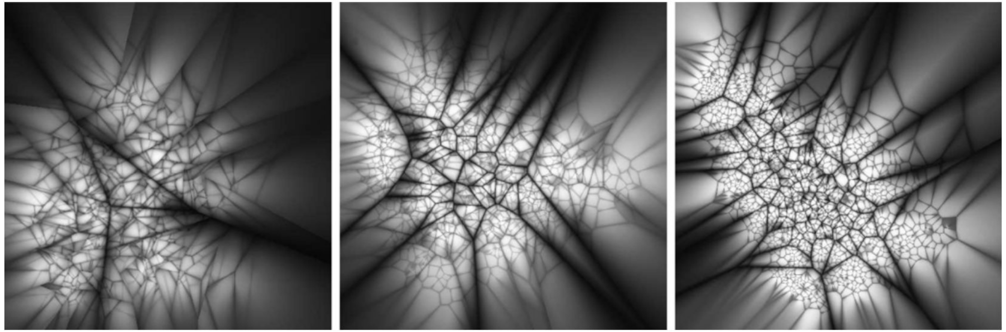
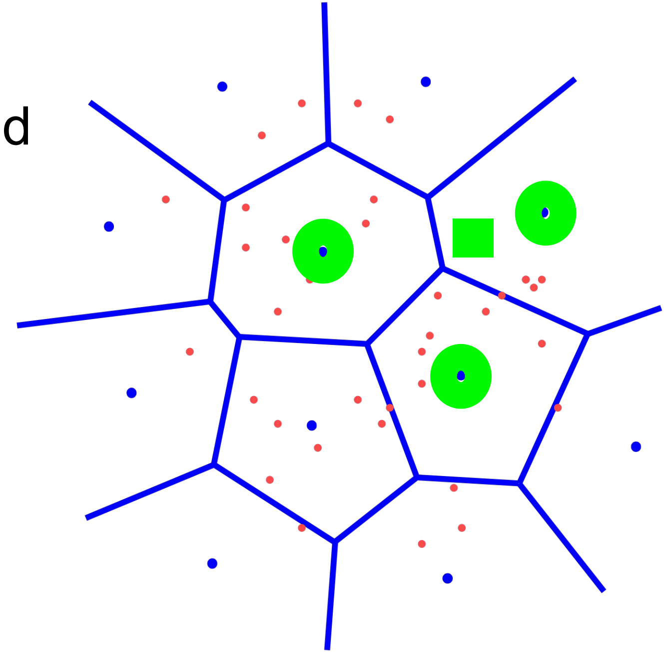


Fig. 3. Projections of priority search k-means trees constructed using different branching factors: 4, 32, 128. The projections are constructed using the same technique as in [26], gray values indicating the ratio between the distances to the nearest and the second-nearest cluster centre at each tree level, so that the darkest values (ratio $\approx$ 1) fall near the boundaries between k-means regions.

Vector quantization for search: The inverted file

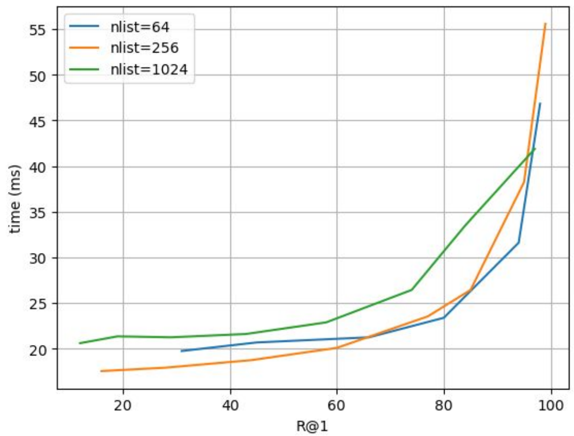
The Inverted File

- Cluster the space into k clusters of vectors
  - assign vectors to nearest centroid
  - Aka. “coarse quantizer”
- index = inverted list structure
  - maps centroid id → list of vectors assigned to it
- search procedure:
  - 1. find np << k nearest centroids to query vector
  - 2. scan the lists corresponding to the np centroids
- Objective: reduce the number of distance computations!

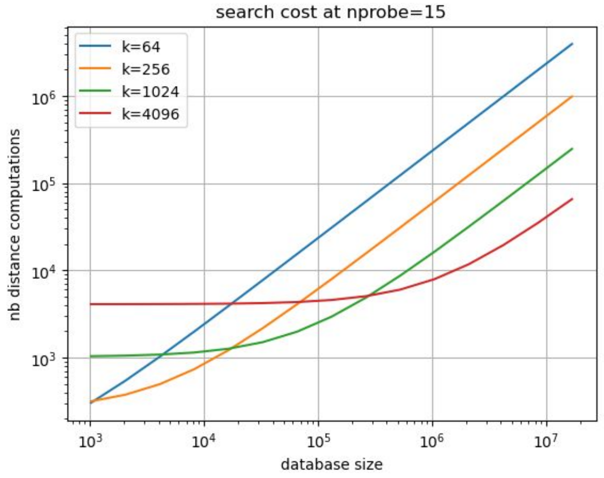


The Inverted File: number of centroids

- Tradeoffs
  - For high-accuracy regime it is not necessary to be very selective: small nb clusters
  - For low-accuracy regime it is better to filter more with clusters

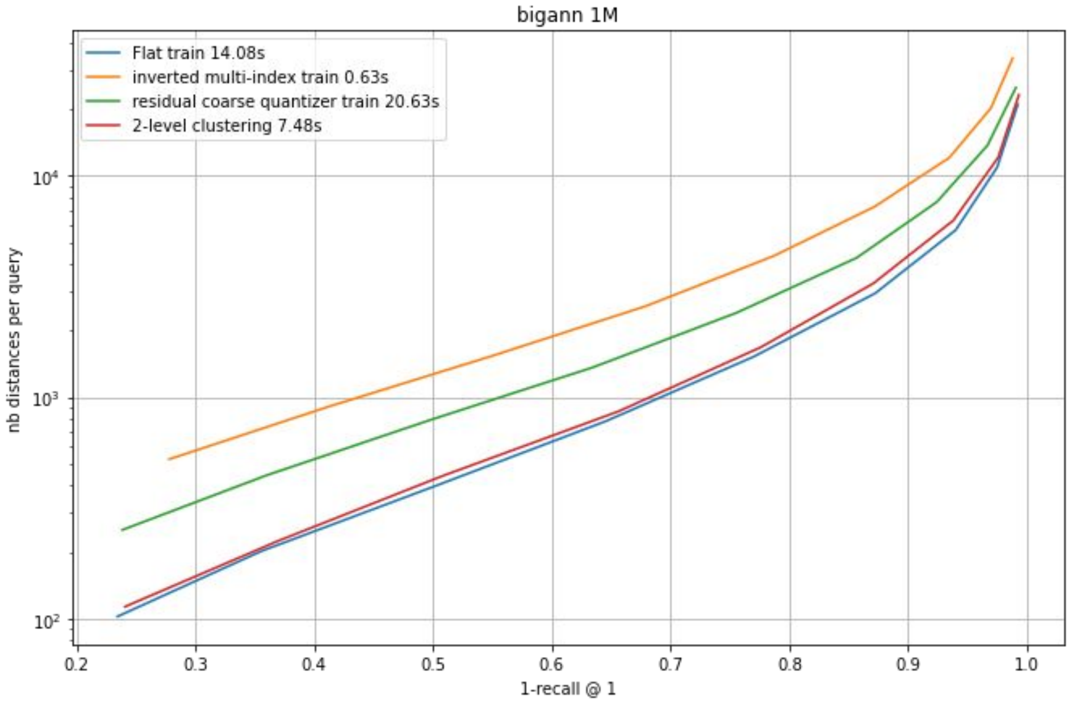


- As a function of database size:
  - Coarse quantization: k distance computations
  - Scanning: (assuming balanced clusters):
    nprobe * n / k
  - Exercise: find optimal k as a function of n for fixed nprobe



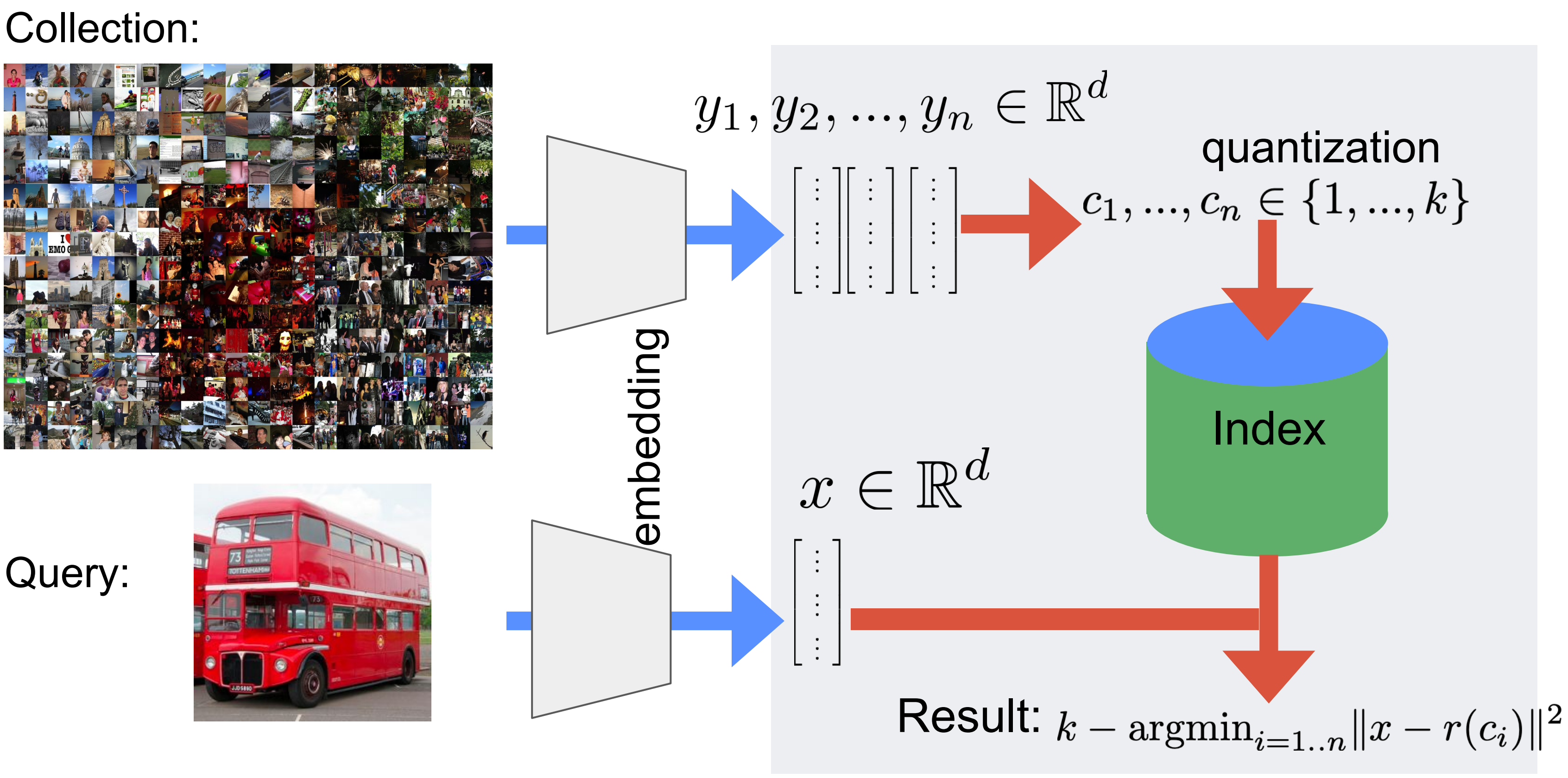
Comparing coarse quantizers

- Useful with 2 levels
  - Reduces complexity



Vector quantization for compression

Compression and search: asymmetric case



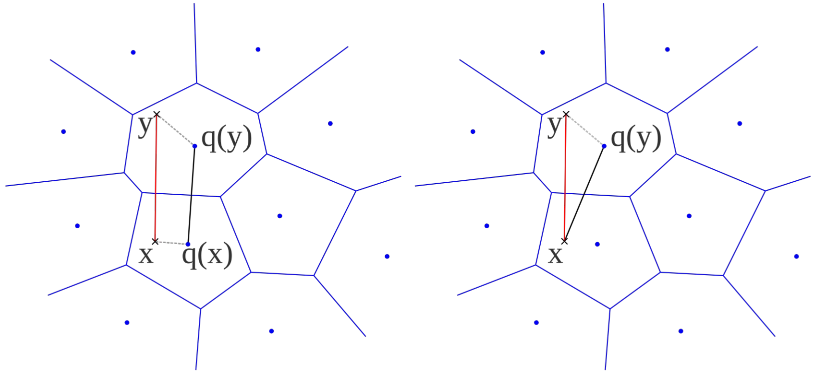
Symmetric vs. asymmetric comparison

- Consider asymmetric setting
  - no constraint on storage of query
  - keep full query vector, encode database vectors $q(y)$
- Distance estimator
  - reproduction value of the quantizer: centroid
  - approximate distance

$$\|x-y\| \approx \|x-q(y)\|$$

  - approximate nearest neighbor

$$\mathrm{argmin}_i \|x-q(y_i)\|$$



symmetric case   asymmetric case

Distance computations with look-up tables

- For a given query x, there are k possible distances
  - Precompute a table!
  - At search time, look up the distances in the table.
  - No computations at search time, only look-ups
  - Useful if nb centroids << nb database elements

- However, this is pretty limited
  - Small codes – limited recall…

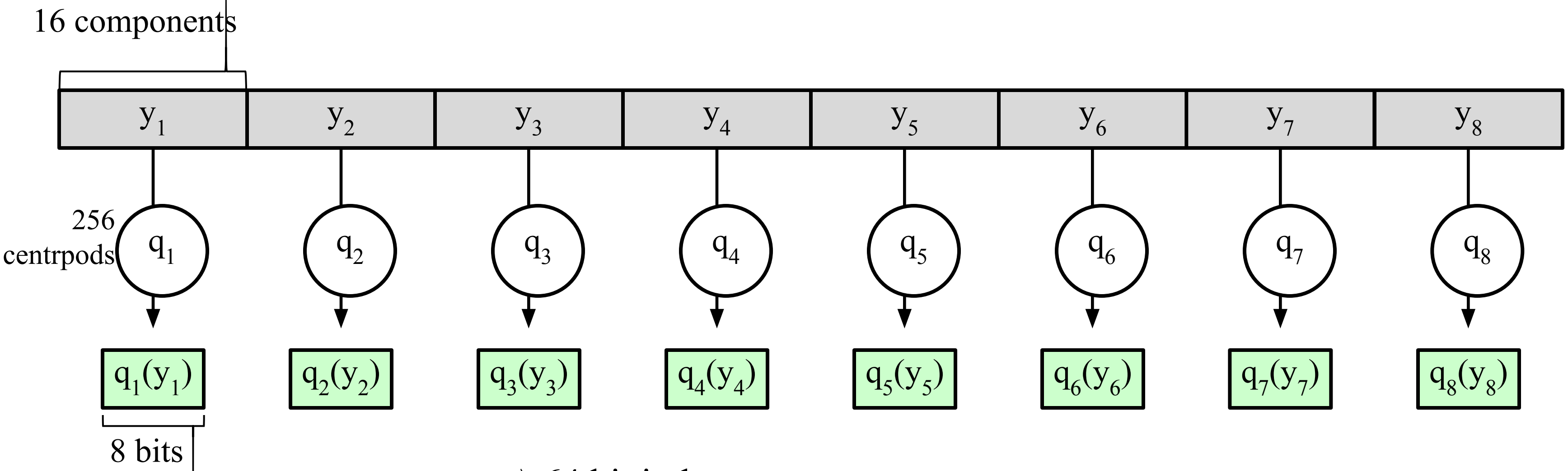
Multi-codebook quantization

Multiple-codebook quantization

- Combine multiple quantizers
  - Each has its codebook
  - Separate codes, total size $\sum_{m=1}^{M} \lceil \log_2(k_m) \rceil$
- In the following:
  - Product quantization
  - Additive quantization

[Jégou & al, Product quantization for nearest neighbor search, PAMI'11]

Product Quantization



$\Rightarrow$ 64 bit index

Multiple-codebook quantization

- reconstruction value: concatenation of centroids

$$y \approx [q_1(y^1), ..., q_m(y^m)]$$

- distance computation: distance is additive

$$\|x - y\|^2 \approx \sum_{j=1}^{m} \|x^j - q_j(y^j)\|^2$$

- precompute *M* look-up tables!

Sizes & flops

	no compression	vector quantizer	product quantizer
code size	d	log2(k)	m*log2(k)
quantization cost	N/A	k*d	k*d
distance computation cost	d multiply-adds	one look-up, one add	m look-ups, m adds
number of distinct values	N/A	k	k^m

[Jégou & al, Product quantization for nearest neighbor search, PAMI'11]

Product Quantizer tradeoffs

- For a given code size
  - Code_size = M * log2(k)
- Higher k (and lower M)
  - Better accuracy
  - Larger quantization tables (slower)

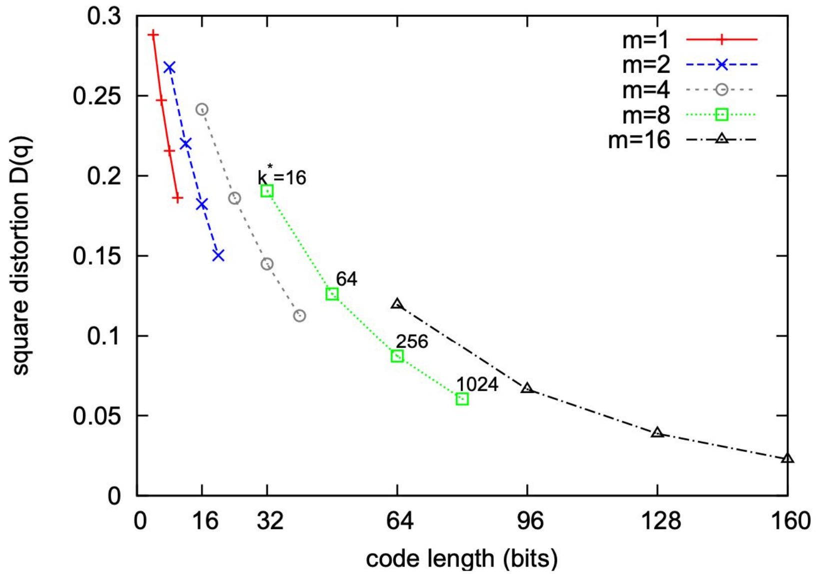


Fig. 1. SIFT: quantization error associated with the parameters m and k^*.

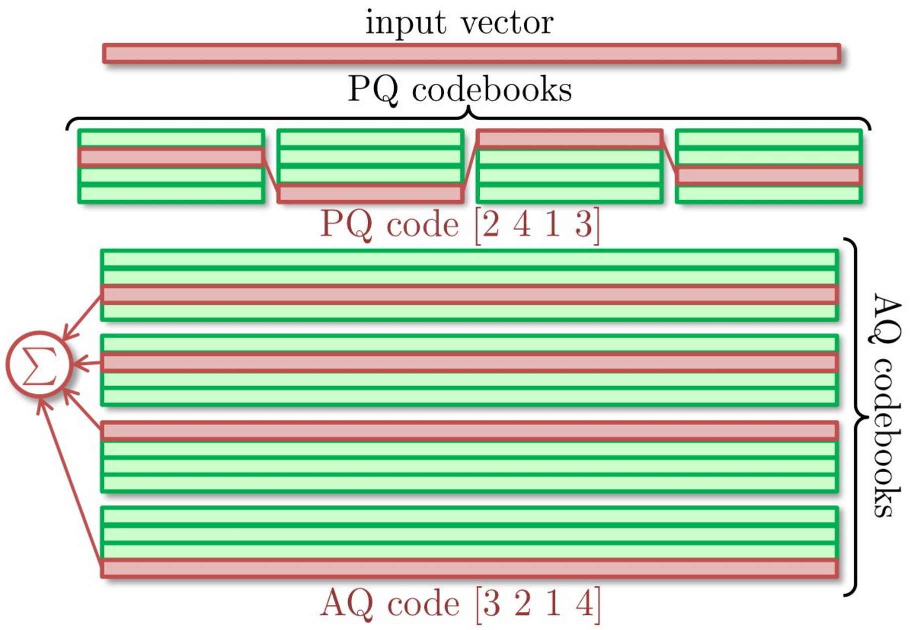
Optimized product quantizat

Additive quantization

[Babenko & Lempitsky,Additive quantization for extreme vector compression, CVPR'14]

Residual quantization

- Use multiple codebooks
  - But in dimension d



General additive quantization – LSQ

- 12x3 h
- + guest talks
- Evaluation via ….

Fast search with additive quantization

- 12x3 h
- + guest talks
- Evaluation via ….

Storing the norms

- 12x3 h
- + guest talks
- Evaluation via ….

Neural quantization methods

The catalyzer

- 12x3 h
- + guest talks
- Evaluation via ….

UNQ

- 12x3 h
- + guest talks
- Evaluation via ….

Polysemous codes

Various decoders

- 12x3 h
- + guest talks
- Evaluation via ….

Equivalence of decoders

- 12x3 h
- + guest talks
- Evaluation via ….

Practical implementation: IVFPQ

Combination of IVF and PQ

- PQ encodes residual

Look-up tables

- Predefine table
-

Parallelization strategies

- Parallelize over queries
- Parallelize intra query
- Coarse and fine quant

END