Large Language Models: Vector embeddings

Nathanaël Fijalkow
CNRS, LaBRI, Bordeaux

Vector embeddings

A ***vector embedding*** is a model which turns inputs into dense vectors, in a way that preserves the information the input contains

This can be used for many things, here are two examples:

Example 1: recommendation systems

We have a dataset of stuff:

- Texts
- Images
- Videos
- …

and we would like to be able to retrieve ***similar*** stuff

Example 2: deduplication

We have a dataset of stuff (typically for a training dataset) and we would like to remove (approximate) duplicates

What is a text embedder?

A Transformer model trained on

- **Masked language modeling**: taking a sentence, we randomly mask 15% of the words in the input then run the entire masked sentence through the model and predict the masked words
- **Next sentence prediction**: the models concatenates two masked sentences as inputs during pretraining. Sometimes they correspond to sentences that were next to each other in the original text, sometimes not. The model then has to predict if the two sentences were following each other or not

Examples of text embedders

- BERT: most popular language model on Hugging Face.
- SBERT: Also known as sentence BERT and sentence transformers
- DistilBERT: A lightweight BERT variant
- RoBERTa: short for “robustly optimized BERT pre training approach”, RoBERTa refined the BERT training procedure to optimize its performance

The notions of similarity

There are many, but two main options:

- **Cosine similarity (IP = Inner Product)**, which becomes dot product if the vectors are normalized
- **Euclidean distance (L2)**

Similarity is different from hashing

Both **hashing** and **similarity search** are about mapping objects to dense vectors, but:

- Hashing is about avoiding collisions for different objects
- Similarity search is about enforcing collisions for similar objects

In practice

A very efficient, well maintained, and SOTA library is FAISS: Facebook AI Similarity Search

Baseline: exhaustive search

We have a dataset of N vectors and a query vector

Exhaustive search computes similarity scores between the query and each vector in the dataset:

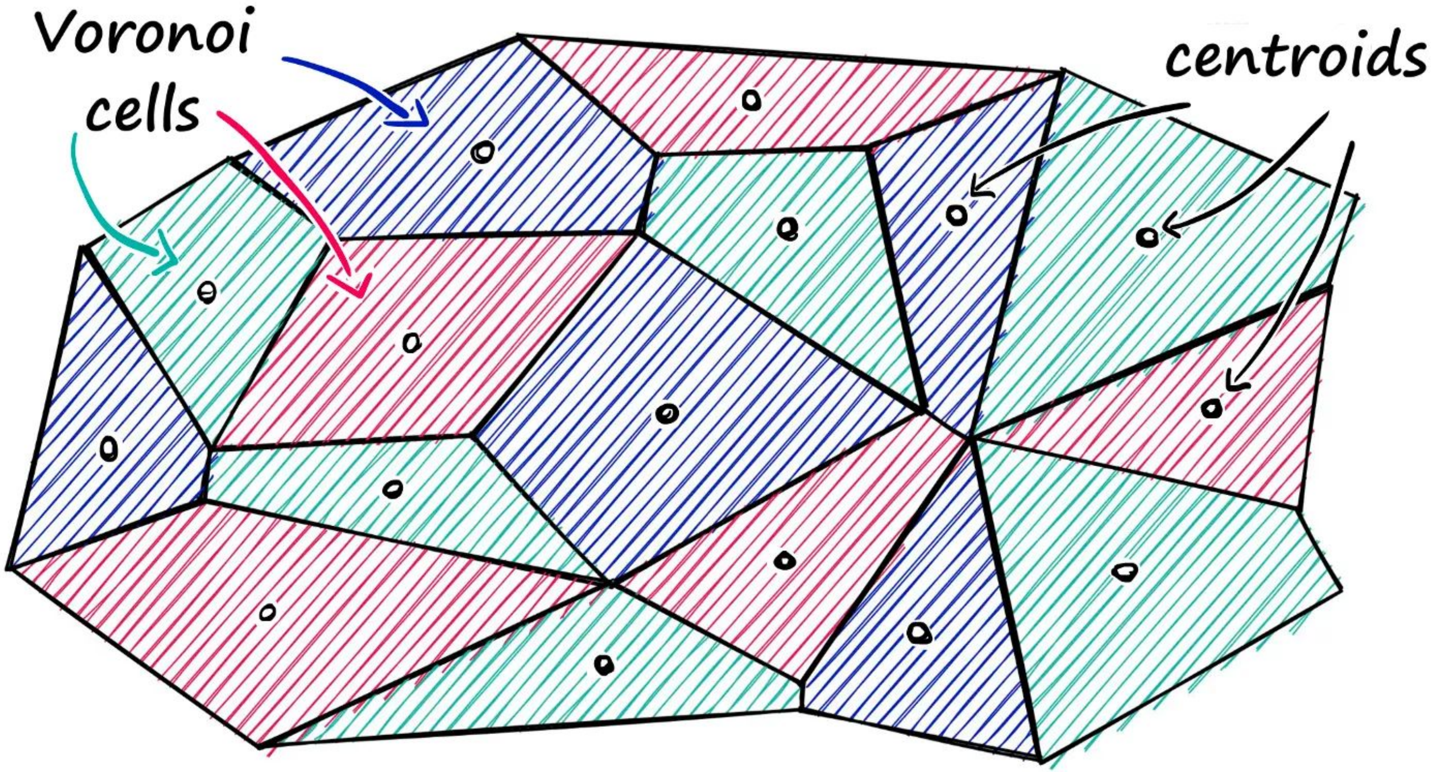
- Very accurate
- But slow and heavy on memory

IndexFlatL2 and *IndexFlatIP* implement exhaustive search

First optimization: Inverted File Index

Inverted File Index (IVF) reduces search space through clustering:

First optimization: Inverted File Index



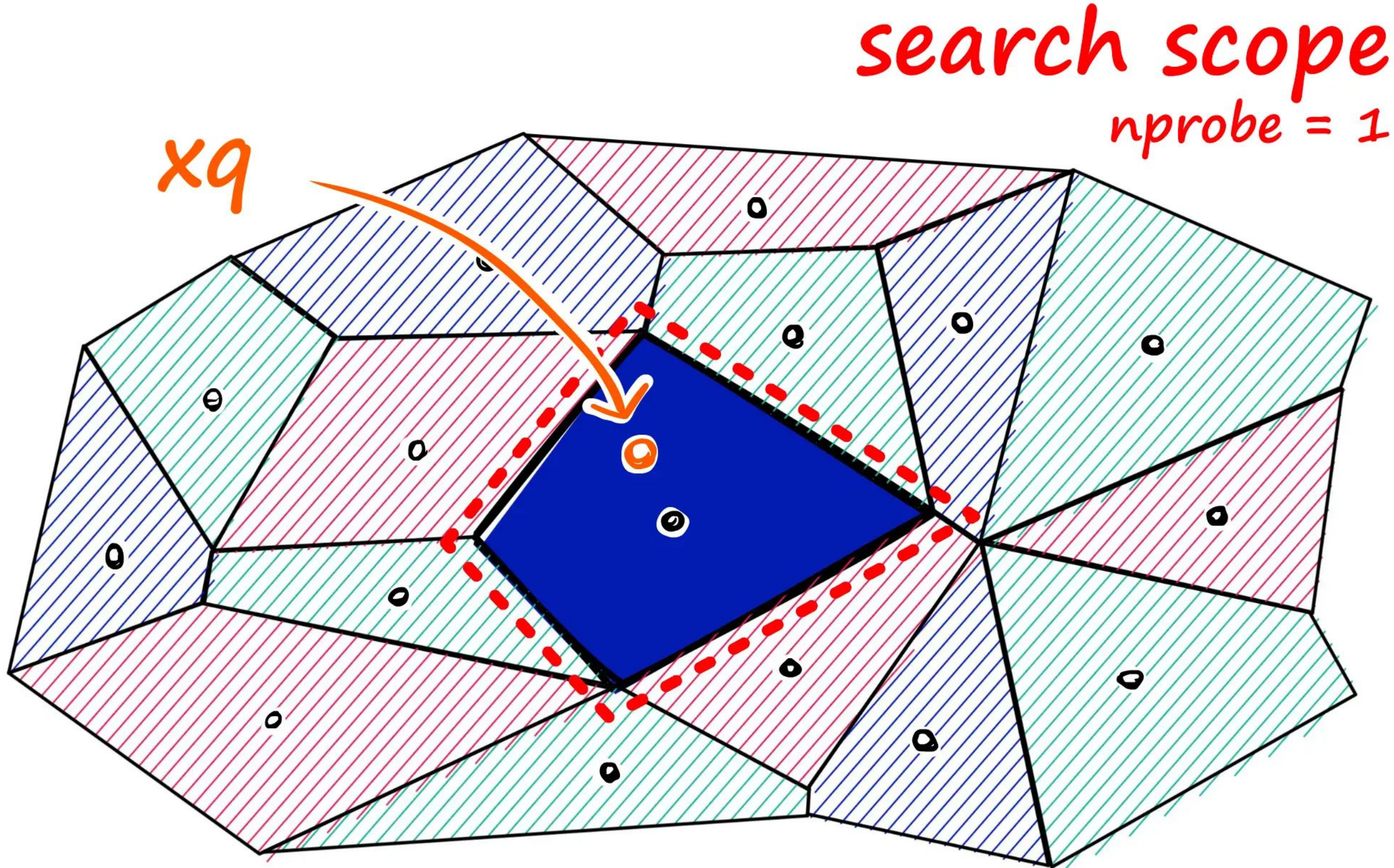
First optimization: Inverted File Index

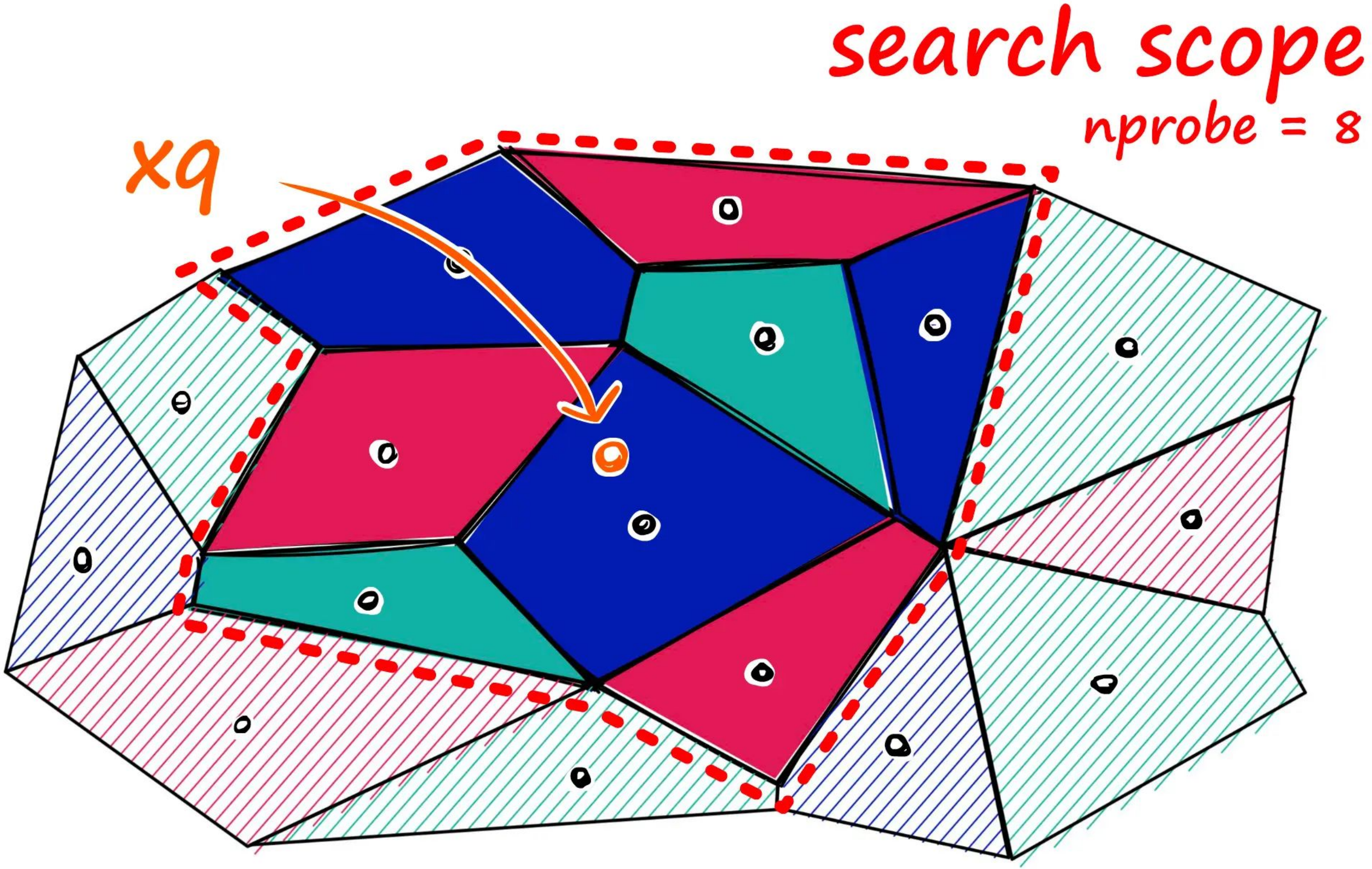
Key idea: we compare the query vector first only to centroids (*nlist*), pick a number of cells (*nprobes*), and then to all vectors in the selected cells

Important: gives approximate results, but a lot faster and less memory. Two parameters:

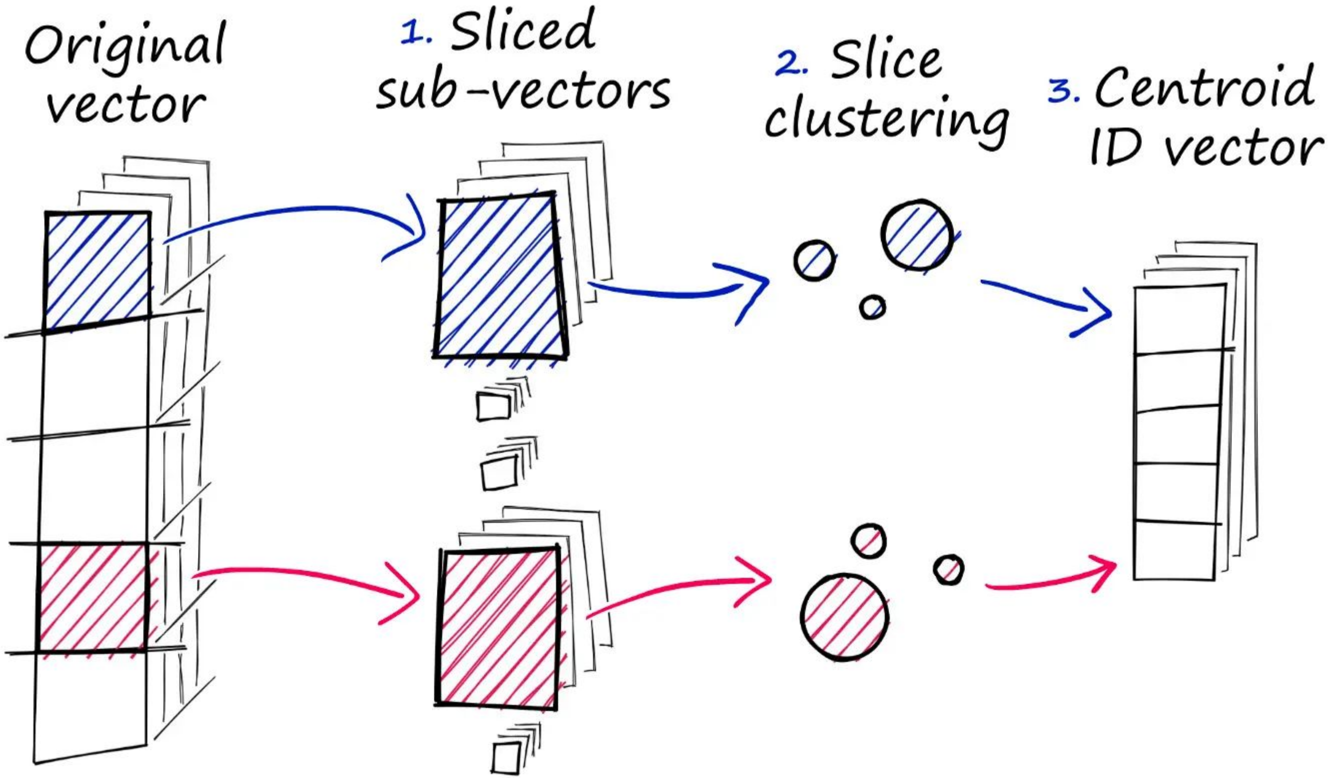
- Number of centroids (*nlist*)
- Number of cells considered (*nprobes*)

IndexIVFFlat implements the index partition


search scope
nprobe = 1
xq


search scope
nprobe = 8
xq

Second optimization: product quantization



Second optimization: product quantization

Objective: approximating similarity score computation

Key idea:

- We split a vector in dimension d into m “subvectors” of dimension d/m
- In each subspace of dimension m we run a clustering algorithm
- We represent a subvector by the (id of the) closest centroid

IndexPQ implements both optimizations

Two optimizations combined

Key idea:

- First, an IVF index to reduce to a small number of cells
- Then, a PQ index in a much smaller set of vectors

IndexIVFPQ implements both optimizations

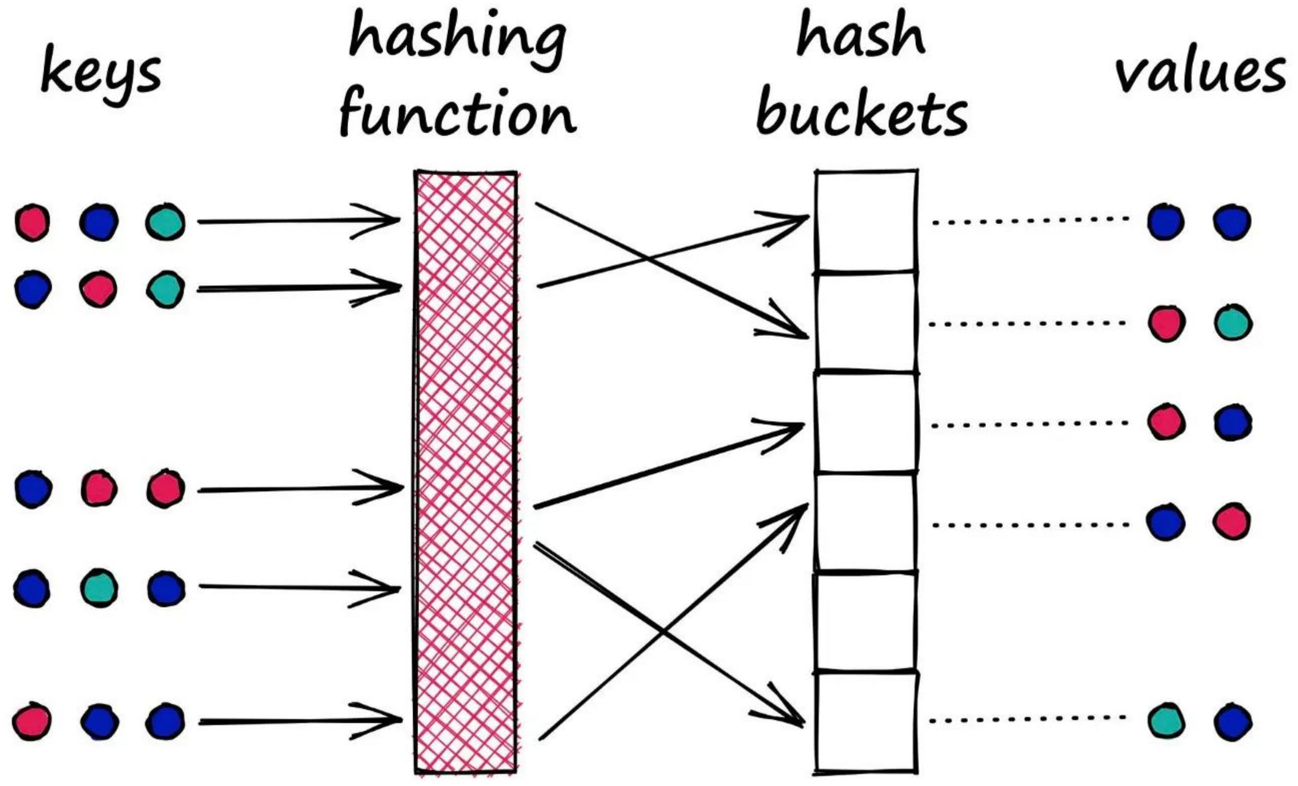
Locality sensitive hashing

Two step process

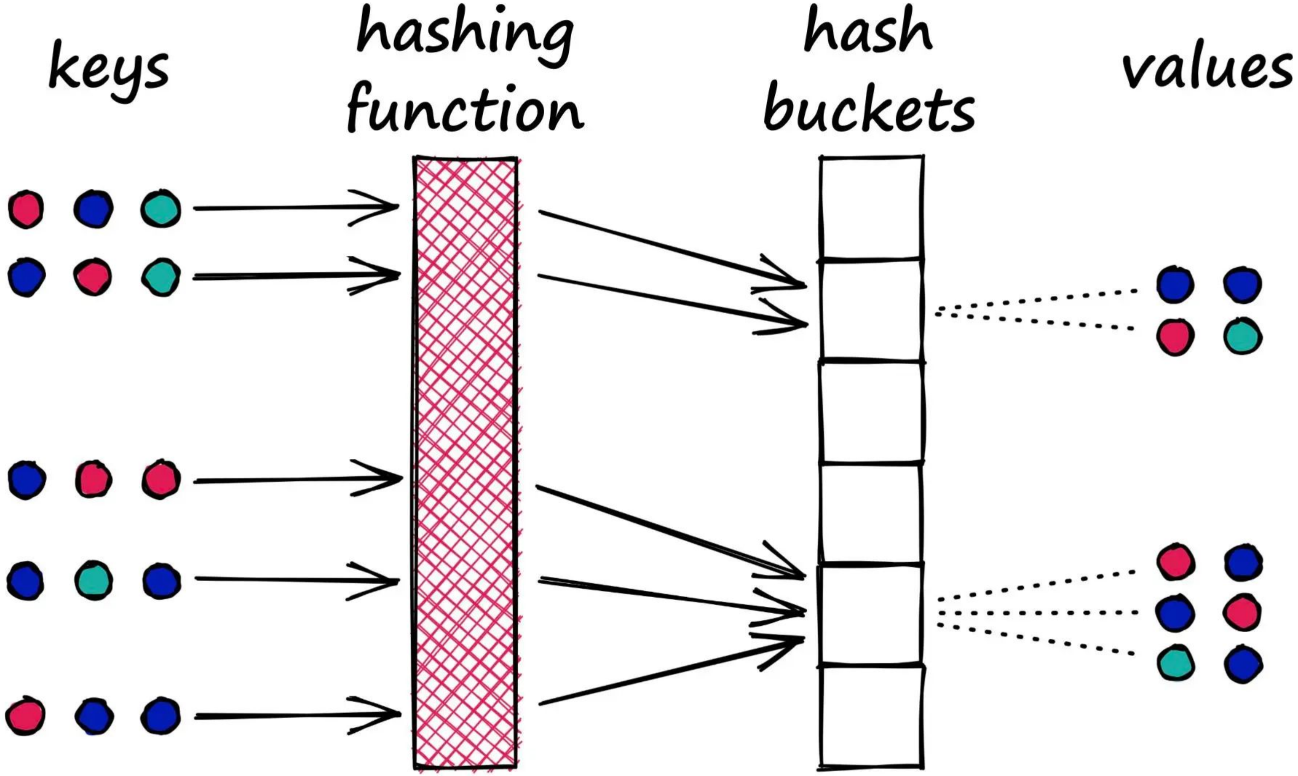
Step 1: identify candidate pairs

Step 2: exhaustive search on candidates

Hashing: minimising collisions



LSH: MAXIMISING COLLISIONS FOR SIMILAR INPUTS

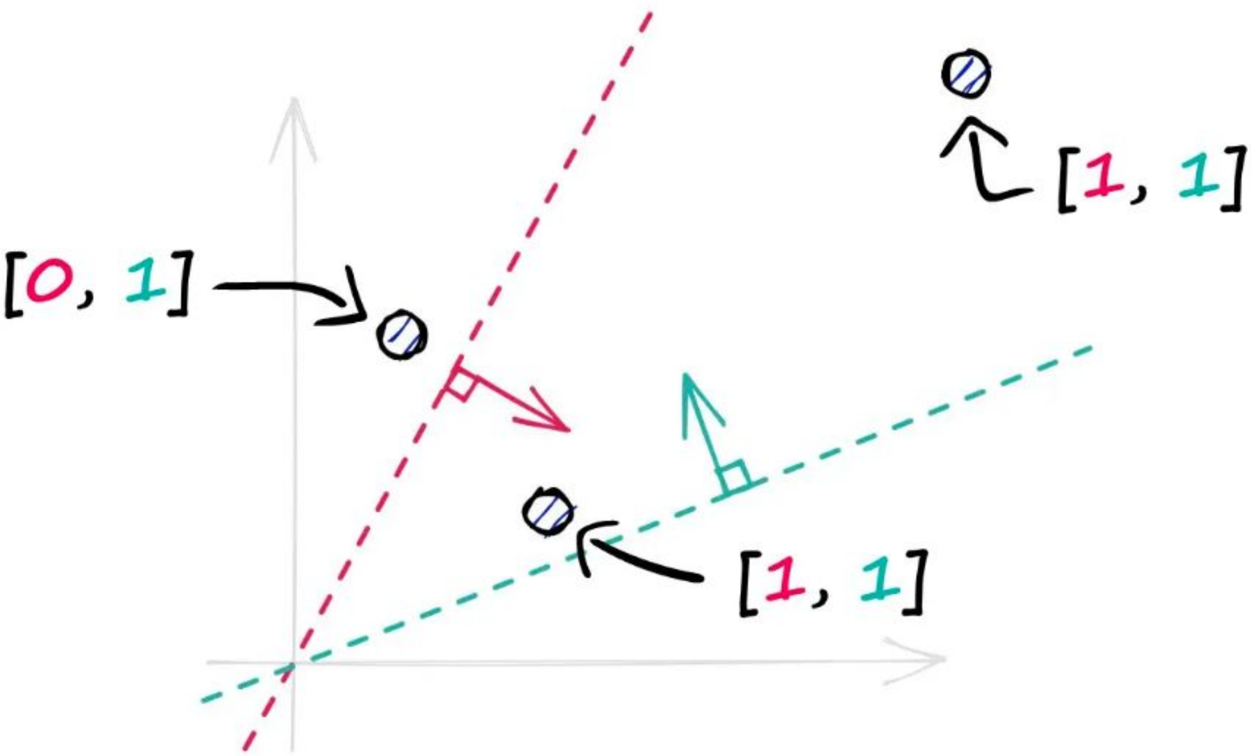


LSH is different from hashing

There are two main approaches:

- Random projections
- Shingling, MinHashing, Banding

LSH with random projections

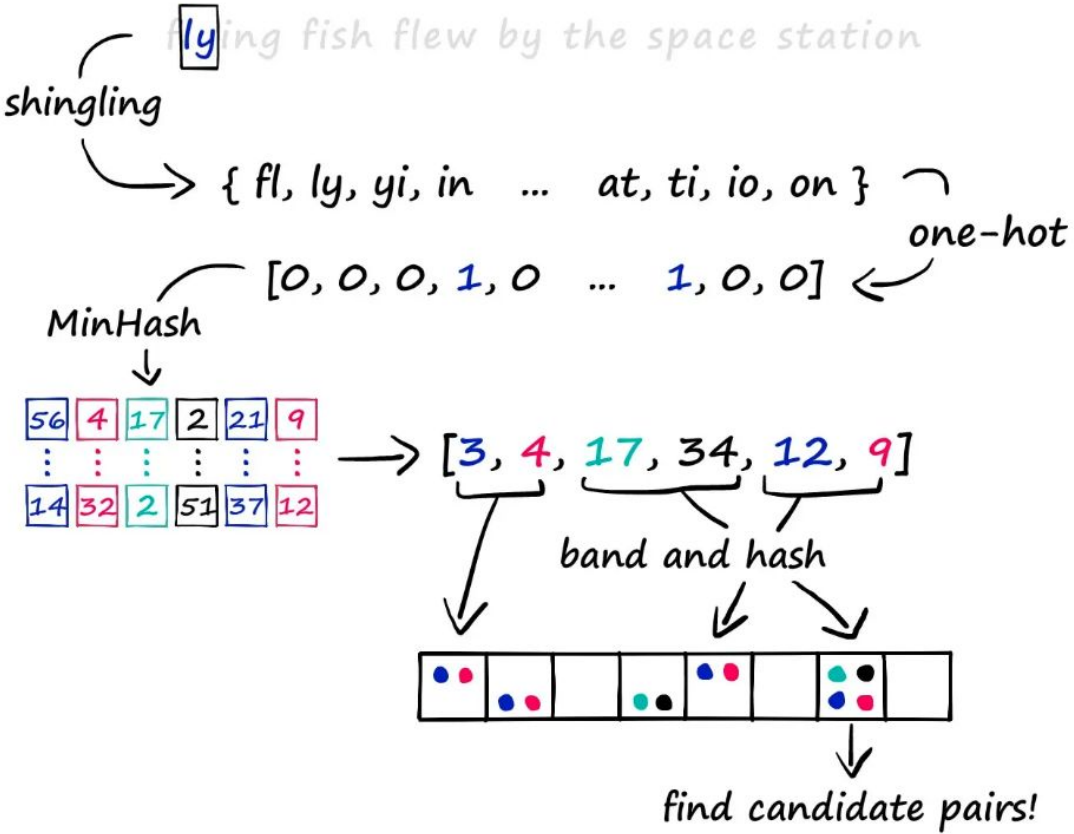


LSH with random projections

Key idea:

- We define *nbits* random hyperplanes in embedding space, each represented by the normal vector
- We represent a vector by the boolean vector expressing on which side of each hyperplane the vector falls on, determined by the sign of dot-product
- We compute Hamming distance between boolean vectors to determine similarity

LSH with shingling, MinHashing, banding



A great reference

[Faiss: The Missing Manual]