

# *Course Project: Product quantizer for NN Search*

Lecturer: Dr. Wan-Lei Zhao

January 13, 2026

## 1 Problems

- You are required to implement product quantizer (without inverted file support) to undertake the asymmetric NN search that is described in [1]. You are already provided with most of the codes for the project. You only need to digest the provided codes and fill the missing parts. Specifically, you are required to fulfill the following steps.
  1. Fill the missing part in “pqencoder.cpp” and “pqnnsearch.cpp”;
  2. Perform product quantization on vectors by calling member function “test()” in class **PQEncoder**;
  3. Perform nearest neighbor search by calling member function “test()” in class **PQnnSearch**;
  4. When you perform the nearest neighbor search in PQnnSearch, you are provided with the code to do performance evaluation by measuring top-1 recall at first **k** ground-truth neighbors;
  5. You are required to show the performance of your implementation by trying different **k**. Please explain why the results are like this and any potential way you can figure out to further improve the NN search quality or speed.

In the provided codes, the part that needs to be completed by you has been hinted by comments as follows. You are NOT suggested to modify the other codes unless you fully understand how they work.

## 2 Checklists of Available Materials

1. C++ codes
  - (a) “abstractpqquantizer.h” and “abstractpqquantizer.cpp” are abstract class for both PQ encoder and PQ based search
  - (b) “pqencoder.h” and “pqencoder.cpp” in charge of product quantizer on the input vector dataset
  - (c) “pqnnsearch.h” “pqnnsearch.cpp” in charge of NN search with asymmetric product quantization
  - (d) “vstring.h” and “vstring.cpp” are responsible for string related operations
  - (e) “evaluator.hpp” in charge of NN search performance evaluation
  - (f) A cmake file that is used to build the project “CMakeLists.txt”
2. Datasets

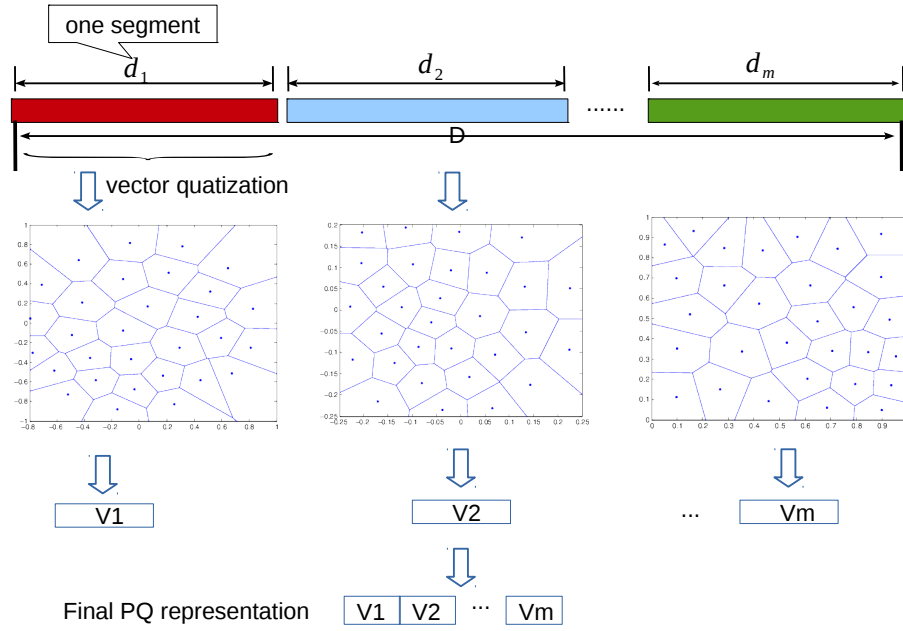


Figure 1: PQ Encoding

- You are provided with one million-level dataset, including vectors file (“fvecs”), query files (“fvecs”), the search ground-truth (“txt”), and a product-quantizer codebook with 8 segments, namely “vocab\_pqk256m8.txt”.
- The links to download the datasets can be found from the attached text file “[links2data.txt](#)”

### 3 Instructions

**Compile the project** The project is fully developed with C++. One is recommended to compile and run it under Linux (Ubuntu in particular) environment. The project can be compiled smoothly by following commands given below.

```
1 cd project2025/tests/
2 cmake .
3 make
```

**Implement and Run PQ encoder** In class **PQEncoder** (file “pqencoder.cpp”), you are required to complete the code in **PQEncoder::quantzPQ**, which is listed below. The general procedure of PQ encoding is illustrated in **Fig. 1**.

```
1 for(s = 0; s < nSeg; s++)
2 {
3     p_pqVocab = this->pqVocab + s*this->pqDim*this->pqNum;
4     locs      = s*this->pqDim;
5     mdst      = RAND_MAX;
6     nnIdx      = 0;
7     for(i = 0; i < this->pqNum; i++)
8     {
9         ppq = p_pqVocab + i*this->pqDim;
10        dst = 0;
```

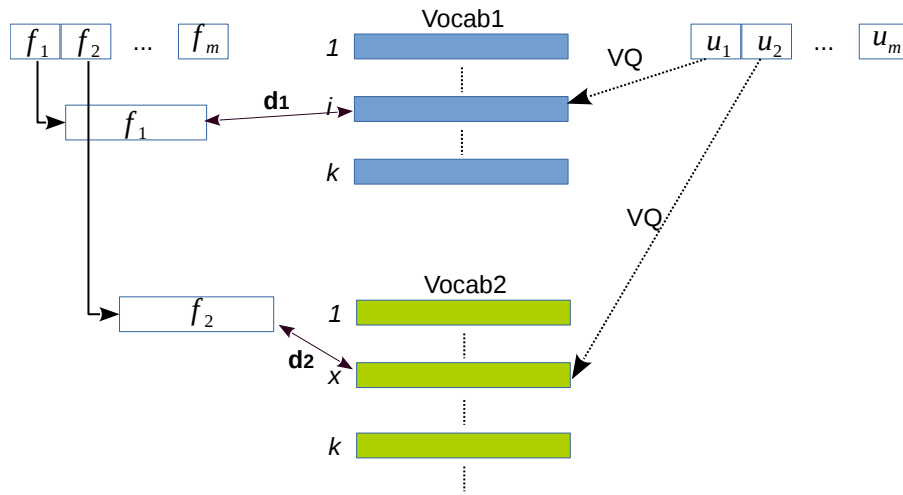


Figure 2: Asymmetric PQ search

```

11      /**
12      *filling your code here
13      */
14  }
15  codes[s] = nnIdx;
16 }

```

quantzPQ()

Given you already implement the PQ encoder, you can call the encoder in “main.cpp” with following modification. Before you run that, you should specify the paths for source file “srcFn”, destine file “dstFn”, and codebook/vocabulary file “vocabFn” in function **Encoder::test()** properly.

```

1 int main(int argc, const char *argv[])
2 {
3     //testNNSearch();
4     testEncode();
5     return 0;
6 }

```

Given the above modifications are done, you can run “make” under “tests/” again. Then you can run the PQ encoding. The code will generate PQ codes in pure text format. One line keeps the PQ codes for one vector. You are provided with a “sample.fvecs” and its corresponding PQ code file. **If your implementations are correct, you should build the same PQ codes as the “sample\_pq.txt” when “sample.fvecs” is used as the source file.**

**Implement and Run PQ-based NN search** The general procedure of asymmetric PQ-based NN search is illustrated in **Fig. 2**. You are required to implement asymmetric PQ-based NN search in class **PQnnSearch** (file “pqnnsearch.cpp”).

In class **PQnnSearch**, you are provided with **PQnnSearch::updateADCTab**, which is listed below. So please fill the inner part of the function.

```

1     for(s = 0; s < nSeg0; s++)
2     {

```

```

3      p_pqVocab = this->pqVocab + s*this->pqDim*this->pqNum;
4      locs      = s*this->pqDim;
5      for(i = 0; i < this->pqNum; i++)
6      {
7          ppq = p_pqVocab + i*this->pqDim;
8          dst = 0;
9          /**
10         *filling your code here
11         */
12         ti = s*this->pqNum + i;
13         this->ADCTab[ti] = dst;
14     }
15 }

```

#### updateADCTab

In class **PQnnSearch** (file “pqnnsearch.cpp”), you are also required to complete the code in **PQnnSearch::performADCQuery**, which is listed below.

```

1  for(ri = 0; ri < this->imgNum; ri++)
2  {
3      dist      = 0;
4      k         = ri*this->nSeg0;
5      ptcodes   = &(refPQCodes[k]);
6      /**
7       *filling your code here
8       */
9      topRank.emplace(pair<float, unsigned>(dist, ri));
10     if(topRank.size() > topk)
11     {
12         topRank.pop();
13     }
14 }//for(ri)

```

#### performADCQuery

In order to call your PQ-based NN search, you need to modify the “main.cpp” as follows. In order to allow the NN search run properly, you should specify the PQ encoding file “ref-pqFn” (produced during PQ encoding), the PQ codebook/vocabulary “vocabFn”, the query file “queryFn”, and the NN search ground-truth for this dataset “gtFn”.

```

1  int main(int argc, const char *argv[])
2  {
3      testNNSearch();
4      //testEncode();
5      return 0;
6  }

```

Given the above modifications are done, you can run “make” under “tests/” again. Then you can run the PQ-based NN search. After the search is done, you should get recall above **21.0%**. While when you modify the parameter “checkK” for function “getRecall()”, different performance may be reported.

## 4 Requirements for your Submission

You are required to submit all your source codes (“.hpp” and “.h”) as well as a “Makefile” as a zipped package (e.g., **studnumb.zip**). Source or generated data files **SHOULD NOT BE** submitted. Before your submission, please make sure they can be compiled soundly under Linux. It is strongly recommended that you work with GCC C++ compiler, either you are with Windows or Linux. The g++ version is assumed to be **5.4** or higher.

Along with the source code, you are required to submit a report written in English. Basically, there should be four sections. In the first section, the general issue that you are going to address should be reviewed. In the second section, please elaborate how you implement the product quantization (both encoder and NN search). In the section followed, please present the performance of your approach on the SIFT1M. Twenty points are set aside for the implementation. Another 10 points are set for the report. Any forms of plagiarism are **PROHIBITED**.

1. Submission due: **2026/Feb./10, 24:00pm**
2. Submit to stonescx@gmail.com
3. Submission should include the source codes that can be compiled smoothly under Ubuntu, and a PDF report

## References

- [1] Hervé Jégou and Matthijs Douze and Cordelia Schmid, “Product Quantization for Nearest Neighbor Search,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, 2011, pp. 117–128.