Building a Semantic Search Engine with Vectorized Databases

Team Members:

Name	Id	Email
Salma Muhammed Entsar	**9220364**	**salma.hassan03@eng-st.cu.edu.eg**
Abdelrahman Moustafa Goamaa	**9220475**	**abdelrahman.abdeltawab03@eng-st.cu.edu.eg**
Muhammed Khaled Ahmed	**9220681**	**mohamed.abdelwahed03@eng-st.cu.edu.eg**
Youssef Roushdy	**9220985**	**youssef.siha03@eng-st.cu.edu.eg**

Introduction:

Why We Considered **IVF + PQ**

One of the biggest challenges was finding the right balance between memory usage, speed, and accuracy. We know that in later stages we'll be scaling up to larger datasets(1M, 10M, 15M, 20M) rows — possibly reaching millions of vectors, with each vector having 70 dimensions. This means that memory consumption would increase significantly, and the search time would also get much slower if we relied on basic brute-force search. That's why we began researching ways to reduce **storage size** and make the **search faster** without losing too much accuracy. Through our research, we found that **Product Quantization (PQ)** is one of the most effective methods for tackling this problem. PQ compresses high-dimensional vectors — in some cases reducing memory usage by up to 97% — and speeds up nearest-neighbor search by approximately 5 times compared to non-quantized methods.

However, PQ alone isn't enough, because it still requires a way to narrow down the search space quickly.

To address that, we decided to combine it with the Inverted File Index (IVF) approach. This IVF + PQ hybrid approach gives us a strong balance overall:

- It significantly reduces memory usage(PQ).
- It speeds up the search process considerably(IVF).
- It maintains a fairly good recall compared to other methods we looked into, like LSH and HNSW, but with good parameters to improve recall.

Expected Parameters for our project:

Datset_Size	nlist	Nprobs (5–10% of the nlist)	m	nbits
1,000,000	1k c=1	50 – 100	7	8
10,000,000	10k c=3	500 – 1,000	7	8
15,000,000	15k c=5	750 – 1,500	7	8
20,000,000	20k c = 4	1,000 – 2,000	7	8

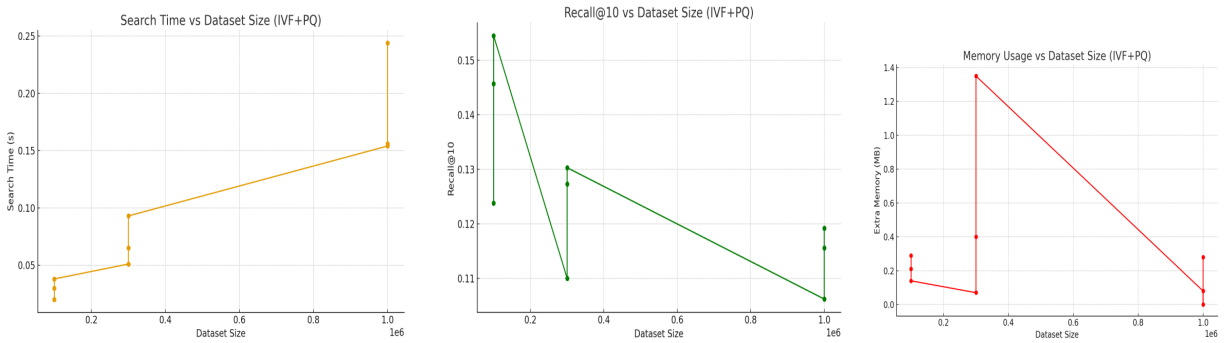
- m: Compression is mainly controlled with the `pq_dim` parameter, and often a good technique for tuning this parameter is to start with one fourth the number of features in your dataset, increasing it in steps 70/4 =18, so m=10,7.
- nlist: The number of clusters (nlist) can be estimated using the rule of thumb nlist = c × √N, where N is the dataset size and c is a scaling constant that depends on data distribution and available memory. Some resources use √N directly, while others multiply it by a small constant c. We plan to experiment with both to find the most suitable configuration for our system constraints.
- Nprobs: Increasing `nprobe` improves recall but also increases query time. To maintain a reasonable trade-off between accuracy and efficiency, we will start with *5% of nlist* and, if time allows, test up to *10%* to explore the effect on retrieval quality and latency.

Experimental Evaluation:

This section presents the experimental evaluation of the FAISS IVF+PQ index structure. We benchmarked performance across different dataset sizes to assess scalability in terms of query latency, recall, and memory consumption.

Dataset_Size	nlist	nprobe	Index_Type	Time_s	Recall@10	Extra_Mem_MB
100,000	316	15	IVF+PQ	0.02	0.1238	0.29
100,000	948	47	IVF+PQ	0.03	0.1457	0.21

Dataset_Size	nlist	nprobe	Index_Type	Time_s	Recall@ 10	Extra_Mem_MB
100,000	1581	79	IVF+PQ	0.038	0.1545	0.14
300,000	547	27	IVF+PQ	0.051	0.11	0.07
300,000	1643	82	IVF+PQ	0.065	0.1273	0.4
300,000	2738	136	IVF+PQ	0.093	0.1303	1.35
1,000,000	1000	50	IVF+PQ	0.154	0.1062	0.08
1,000,000	3000	150	IVF+PQ	0.156	0.1156	0.0
1,000,000	5000	250	IVF+PQ	0.244	0.1192	0.28



Conclusion:

As the dataset size increases, search time grows moderately while recall remains fairly stable around 0.11–0.15. Memory overhead stays below 1.5 MB across all configurations, confirming the efficiency of IVF+PQ for scalable vector retrieval. This demonstrates the feasibility of FAISS IVF+PQ indexing for building a semantic search engine in large-scale environments.

References:

EBR Series — IVFPQ Explained Pt 1
Product Quantization: Compressing high-dimensional vectors by 97%
Accelerating Vector Search: NVIDIA cuVS IVF-PQ Part 2, Performance Tuning
Performance FAQ