GRIP: Capacity-Optimized High-Performance Nearest Neighbor Search for Vector Search Engine

Minjia Zhang and Yuxiong He
Email:[minjiaz,yuxhe]@microsoft.com

Microsoft AI and Research
Evolution of Search

Classic information retrieval is based on *keyword matching* and *user behavior signals*.

Novel search scenarios have emerged:

- Natural language conversation
- Question and answer
- Image/multimedia
- Mobile search
- Product search
- ...

*Deep learning based*
Vectorization and ANN Search

R^d (e.g., d = 128)

N = 1~1000 million points

Doc

Embedding (offline)

NRM

ANN: Tree-based, hashing-based, quantization-based, nn-graph based
Vectorization and ANN Search

$N = 1\sim1000$ million points

$R^d$ (e.g., $d = 128$)

ANN: Tree-based, hashing-based, quantization-based, nn-graph based
Three Metrics to Optimize

- **Recall**: The fraction of top-K retrieved are exact nearest neighbors. Important for getting high-quality results.

- **Latency**: Per-query response time. Must return in response time limit.

- **Memory overhead**: Index size. A crucial factor for large-scale dataset.
Three Metrics to Optimize

Recall = The fraction of top-K retrieved are exact nearest neighbors
Important for getting high-quality results

Latency = Per-query response time
Must return in response time limit

Memory overhead = Index size
A crucial factor for large-scale dataset

Can we design an ANN algorithm to achieve low search latency and high accuracy while being memory-efficient and scalable?
Our Results

**GRIP**: A capacity-optimized ANN algorithm that leverages DRAM and SSDs simultaneously

- **Jointly optimize latency, recall, and memory cost**
  - Fast in-memory and end-to-end search time
  - Significantly reduced memory usage

- **Results**
  - Compared to the SOTA graph-based approach, **GRIP** uses **12--14X** less memory with comparable accuracy and latency
  - Compared to a highly efficient compression-based approach, **GRIP** is **14--23X** faster with higher recall and a similar memory cost
Talk Outline

• Background and Challenges

• Design
  • Memory Efficiency
  • Latency Reduction
  • Accuracy Boost

• Evaluation Results
Existing Approaches

Proximity graphs (HNSW, NSG)

- Graph built over database vectors
- Approximates Delaunay graph with great navigability
- N-greedy best-first search until reaching at local optimum

HNSW: https://github.com/nmslib/hnswlib
NSG: github.com/ZJULearning/nsg
Existing Approaches

Proximity graphs (HNSW, NSG)

- Graph built over database vectors
- Approximates Delaunay graph with great navigability
- N-greedy best-first search until reaching at local optimum

✅ High recalls and low latency
❌ Low ratio of vectors/machine
Existing Approaches

Proximity graphs (HNSW, NSG)
- Graph built over database vectors
- Approximates Delaunay graph with great navigability
- N-greedy best-first search until reaching at local optimum

Compression based algorithms (FAISS)
- Compresses vectors into short code to save DRAM
- Combines with two-level index (e.g. IVF) to avoid exhaust search
- At search time, search a few closest clusters

✔ High recalls and low latency
❌ Low ratio of vectors/machine

FAISS: https://github.com/facebookresearch/faiss
Existing Approaches

**Proximity graphs (HNSW, NSG)**
- Graph built over database vectors
- Approximates Delaunay graph with great navigability
- N-greedy best-first search until reaching at local optimum

**Compression based algorithms (FAISS)**
- Compresses vectors into short code to save DRAM
- Combines with two-level index (e.g. IVF) to avoid exhaust search
- At search time, search a few closest clusters

- High recalls and low latency
- Low ratio of vectors/machine

- High ratio of vectors/machine
- Low recall@1, decent recall@100
Existing Approaches

Proximity graphs (HNSW, NSG)
- Graph built over database vectors
- Approximates Delaunay graph with great navigability
- \( N \)-greedy best-first search until reaching at local optimum

Compression based algorithms (FAISS)
- Compresses vectors into short code to save DRAM
- Combines with two-level index (e.g. IVF) to avoid exhaust search
- At search time, search a few closest clusters

✔️ High recalls and low latency ✔️ High ratio of vectors/machine
❌ Low ratio of vectors/machine ❌ Low recall@1, decent recall@100
Drink from Both Glasses?

Proximity graphs (HNSW, NSG)
- Graph built over database vectors
- Approximates Delaunay graph with great navigability
- N-greedy best-first search until reaching at local optimum

Compression based algorithms (FAISS)
- Compresses vectors into short code to save DRAM
- Combines with two-level index (e.g. IVF) to avoid exhaust search
- At search time, search a few closest clusters

Can we get the best of both worlds?

- High recalls and low latency
- High ratio of vectors/machine
- Low ratio of vectors/machine
- Low recall@1, decent recall@100
Design of GRIP
“GRIP” Overview

- **Preview stage (DRAM)**
  - A copy of compressed vectors
  - Graph routing index
  - Memory-bandwidth optimized PQ

- **Validation Stage (SSDs)**
  - A copy of full-precision vectors
  - Lightweight recomputation in full-precision
Memory Efficiency

• **Challenges:** Memory capacity becomes a scalability bottleneck as the #vectors grows
  • E.g., HNSW takes 51G to index 80 millions of 128d feature vectors, which do not fit in 32G main memory

Index size = #vectors x bytes per vector + index metadata

![Bar chart showing index size vs number of vectors](chart.png)

Does not fit into memory

17
Memory Efficiency

• Solution: Graph routing index + Compressed short code
  • Generalized proximity graph as an index of grouped vectors
  • Each group consists of a small set of vectors, compressed with product quantization
  • Graph edge adjusted to provide reachability guarantee
Latency Reduction

• **Challenge:** Distance estimation between the unquantized q and quantized short code can be slow

\[
\frac{\|q - c\|^2}{\text{Term-A}} + \sum_{m=1}^{M} \|c_{x_m}^m\|^2 + 2 \sum_{m=1}^{M} \langle c^m, c_{x_m}^m \rangle - 2 \sum_{m=1}^{M} \langle q^m, c_{x_m}^m \rangle
\]

Asymmetric distance estimator: **2 X M look-ups** to compute similarity score -> **memory bandwidth bound**
Latency Reduction

• **Solution:**
  - Memory bandwidth optimized PQ
  - Estimates distance with *M lookups*: effectively cutting the memory bandwidth consumption by half

Pre-compute partial-distance value (PDV) offline and trade-off 1-float per vector for *M* lookups
Accuracy Boost

• **Challenge:** Vector compression provides memory compactness but results in poor recall on large datasets

Vectors \((y_1, y_2)\) quantized to the same short code \((c_1)\) have the same estimated distance to \(q\)
**Accuracy Boost**

• **Solution:** Keep a copy of full-precision vectors on SSDs and validate a short candidate list from in-memory search.

<table>
<thead>
<tr>
<th></th>
<th>DRAM</th>
<th>SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Cost</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Power consumption</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Scalability</td>
<td>GBs per DIMM</td>
<td>TBs per PCIe</td>
</tr>
<tr>
<td>Latency</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>
Accuracy Boost

- **Challenge:** Accessing SSD is still much slower than accessing DRAM
- **Solution:** Lightweight validation
  - Parallel access multiple flash memory packages to reach high-aggregate bandwidth
  - Hide high SSD latency through overlapping distance computations and IO
Evaluation
Evaluation Metrics

• Recall
• Latency
• Memory cost
• $VQ = V \times Q$

$$= \frac{\#Vectors}{Machine} \times \text{Query processing rate}$$
Evaluation Metrics

• Recall
• Latency
• Memory cost
• $VQ = V \times Q$

$$\text{#Machines} = \frac{\text{#Vectors}}{\text{Machine}} \times \text{Query processing rate}$$

The higher the $VQ$, the less number of machines needed!
Performance: GRIP vs FAISS/IVFPQ

- To get high recall under similar memory cost
  - GRIP is 2–19X faster
  - GRIP improves VQ by 2–12X

- To get similar recall or higher recall target
  - GRIP is 14–23X faster
  - GRIP improves VQ by 12–14X

<table>
<thead>
<tr>
<th></th>
<th>IVFPQ</th>
<th></th>
<th>GRIP</th>
<th></th>
<th>VQ Impr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFTM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>0.679</td>
<td>8.8</td>
<td>40</td>
<td>0.904</td>
<td>0.6</td>
</tr>
<tr>
<td>512</td>
<td>0.678</td>
<td>5.4</td>
<td>40</td>
<td>0.989</td>
<td>1.8</td>
</tr>
<tr>
<td>256</td>
<td>0.676</td>
<td>3.2</td>
<td>40</td>
<td>0.986</td>
<td>1.2</td>
</tr>
<tr>
<td>64</td>
<td>0.662</td>
<td>2.0</td>
<td>40</td>
<td>0.948</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep10M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>0.602</td>
<td>18.5</td>
<td>302</td>
<td>0.906</td>
<td>1.2</td>
</tr>
<tr>
<td>512</td>
<td>0.601</td>
<td>15.6</td>
<td>302</td>
<td>0.975</td>
<td>5.4</td>
</tr>
<tr>
<td>256</td>
<td>0.599</td>
<td>14.1</td>
<td>302</td>
<td>0.965</td>
<td>3.4</td>
</tr>
<tr>
<td>64</td>
<td>0.580</td>
<td>12.8</td>
<td>302</td>
<td>0.906</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpaceV80M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>0.767</td>
<td>28.1</td>
<td>2552</td>
<td>0.925</td>
<td>1.2</td>
</tr>
<tr>
<td>512</td>
<td>0.765</td>
<td>27.8</td>
<td>2552</td>
<td>0.977</td>
<td>4.7</td>
</tr>
<tr>
<td>256</td>
<td>0.763</td>
<td>27.5</td>
<td>2552</td>
<td>0.971</td>
<td>2.7</td>
</tr>
<tr>
<td>64</td>
<td>0.752</td>
<td>27.2</td>
<td>2552</td>
<td>0.946</td>
<td>1.4</td>
</tr>
</tbody>
</table>
Performance: GRIP vs HNSW

- To get similar accuracy and latency
  - **GRIP** improves VQ by **2.5—15X**
  - **GRIP** reduces the memory cost by **12—14X**

<table>
<thead>
<tr>
<th></th>
<th>HNSW</th>
<th>GRIP</th>
<th>VQ Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recall</td>
<td>efSearch</td>
<td>Latency</td>
</tr>
<tr>
<td>SIFT1M</td>
<td>0.993</td>
<td>1280</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>0.973</td>
<td>320</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>0.947</td>
<td>160</td>
<td>0.3</td>
</tr>
<tr>
<td>Deep10M</td>
<td>0.998</td>
<td>1280</td>
<td>3.1</td>
</tr>
<tr>
<td></td>
<td>0.985</td>
<td>320</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>0.969</td>
<td>160</td>
<td>0.4</td>
</tr>
<tr>
<td>SpaceV80M</td>
<td>0.972</td>
<td>2560</td>
<td>4.1</td>
</tr>
<tr>
<td></td>
<td>0.943</td>
<td>640</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td>0.918</td>
<td>320</td>
<td>0.9</td>
</tr>
</tbody>
</table>
# Cost Comparison

<table>
<thead>
<tr>
<th></th>
<th>IVFPQ (Product quantization)</th>
<th>HNSW (Proximity graphs)</th>
<th>GRIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low search latency</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>High accuracy</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Low memory cost</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
Summary

• **GRIP** leverages *both DRAM and SSDs* simultaneously, without the need to scale out to accommodate large datasets

• *Capacity-optimized* through
  • Memory efficiency improvement
  • Latency reduction
  • Accuracy boost

• Support vector search in Microsoft with great cost reduction
Thank you!
Q&A