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

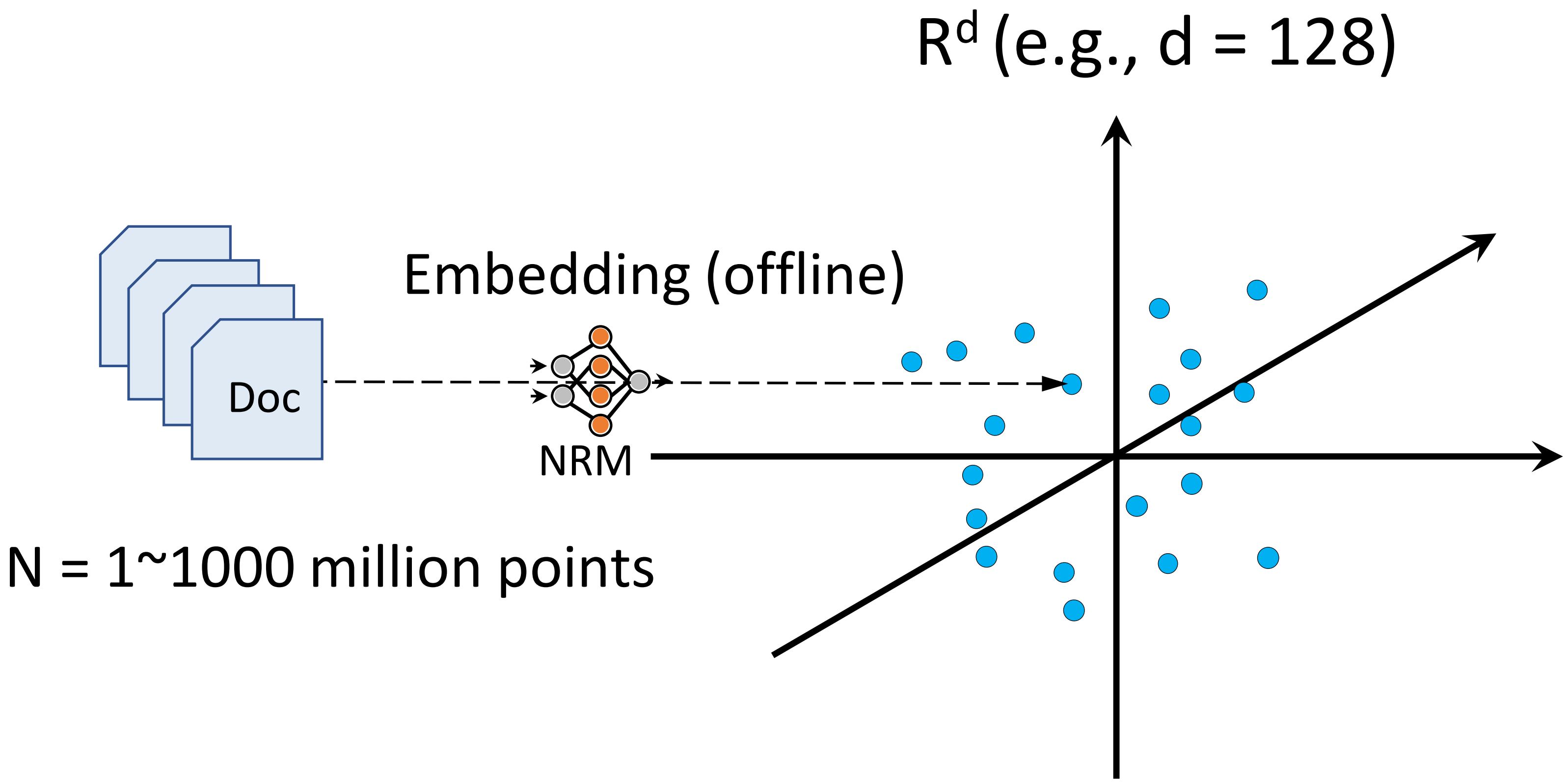
Classic information retrieval is based on *keyword matching* and *user behavior signals*

Novel search scenarios have emerged

Deep learning based

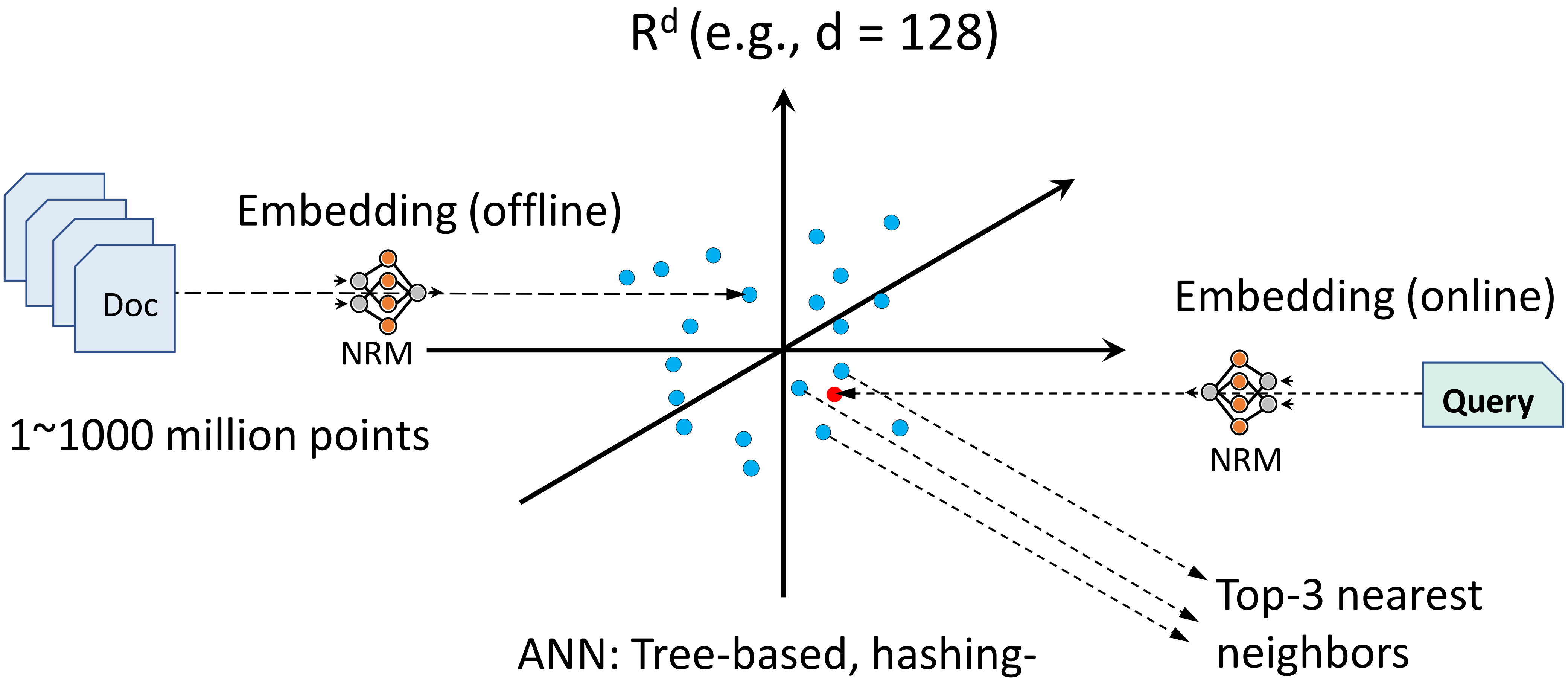
- Natural language conversation
- Question and answer
- Image/multimedia
- Mobile search
- Product search
- …

Vectorization and ANN Search



ANN: Tree-based, hashing-based, quantization-based, nn-graph based

Vectorization and ANN Search



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

<u>Proximity graphs (HNSW, NSG)</u>

- 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

<u>Proximity graphs (HNSW, NSG)</u>

- 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

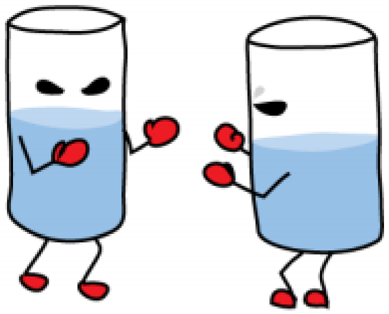
✓ High recalls and low latency

✗ Low ratio of vectors/machine

✓ High ratio of vectors/machine

✗ Low recall@1, decent recall@100

Existing Approaches



<u>Proximity graphs (HNSW, NSG)</u>

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

<u>Compression based algorithms (FAISS)</u>

- 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

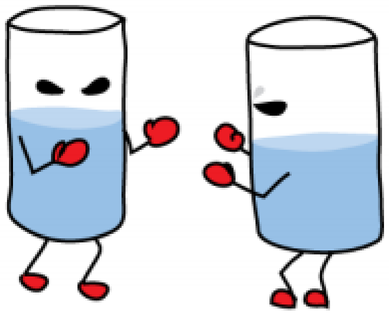
✓ High recalls and low latency

✗ Low ratio of vectors/machine

✓ High ratio of vectors/machine

✗ Low recall@1, decent recall@100

Drink from Both Glasses?



<u>Proximity graphs (HNSW, NSG)</u>

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

<u>Compression based algorithms (FAISS)</u>

- 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

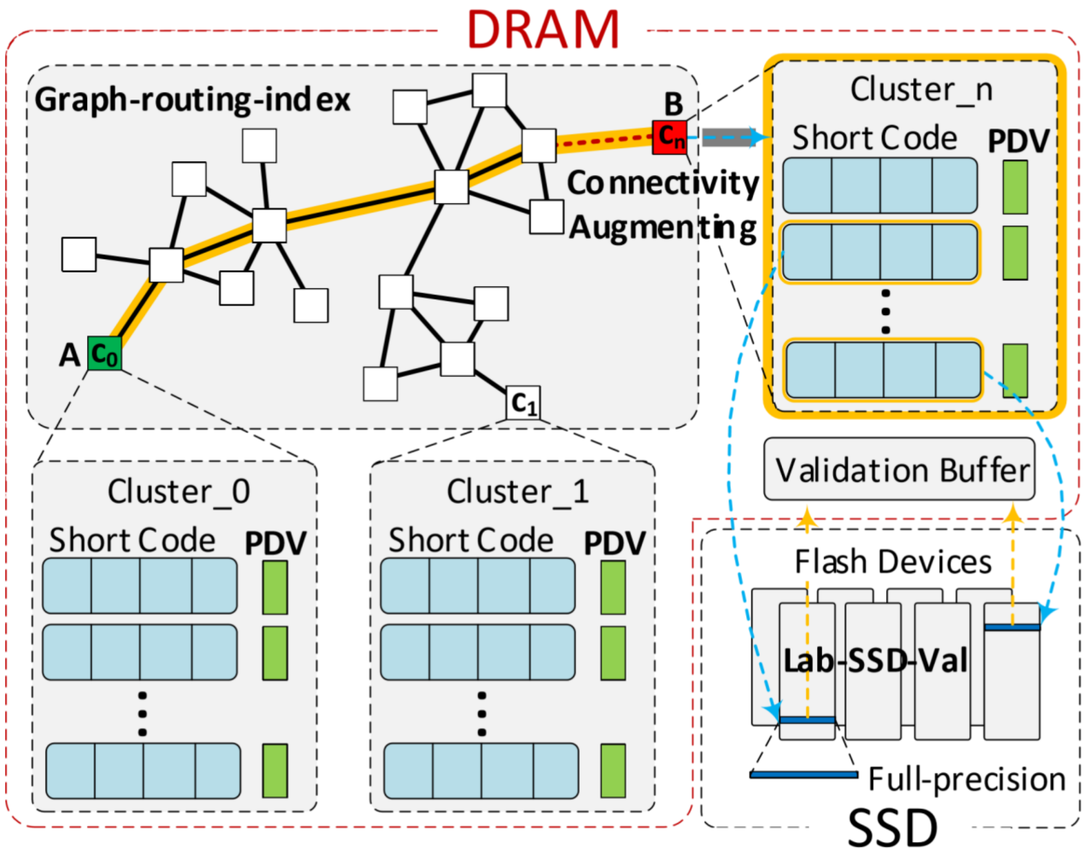
✗ Low ratio of vectors/machine

✓ High ratio of vectors/machine

✗ Low recall@1, decent recall@100

Design of GRIP

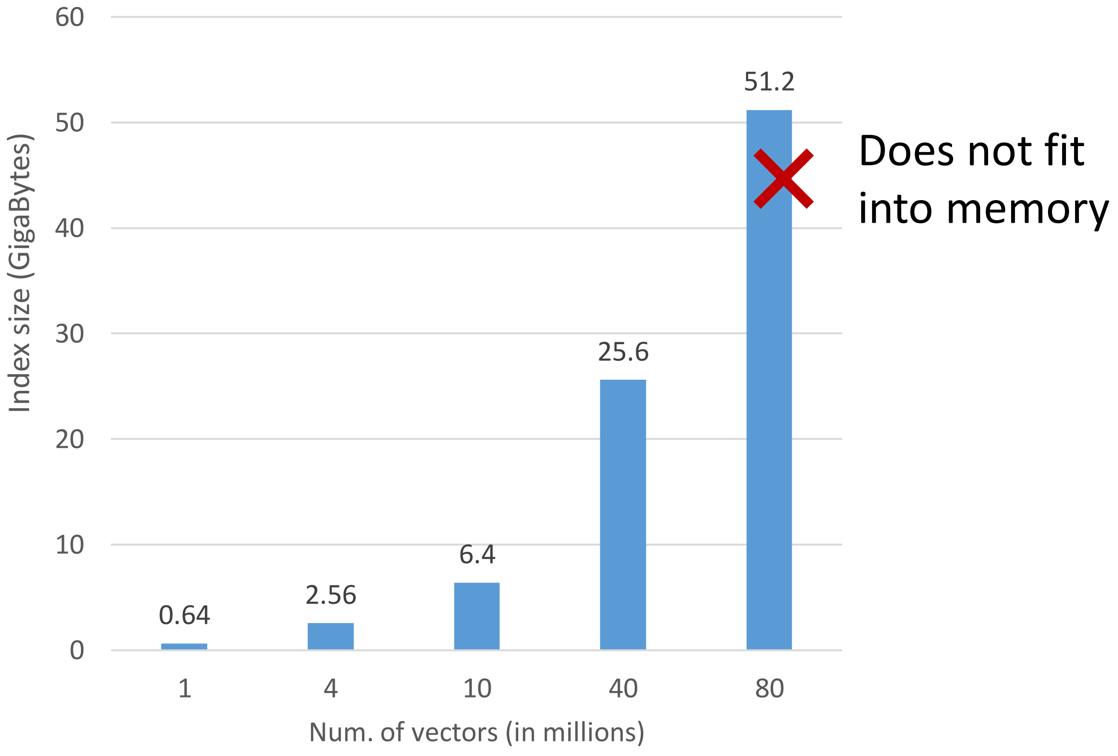
“GRIP” Overview



- 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

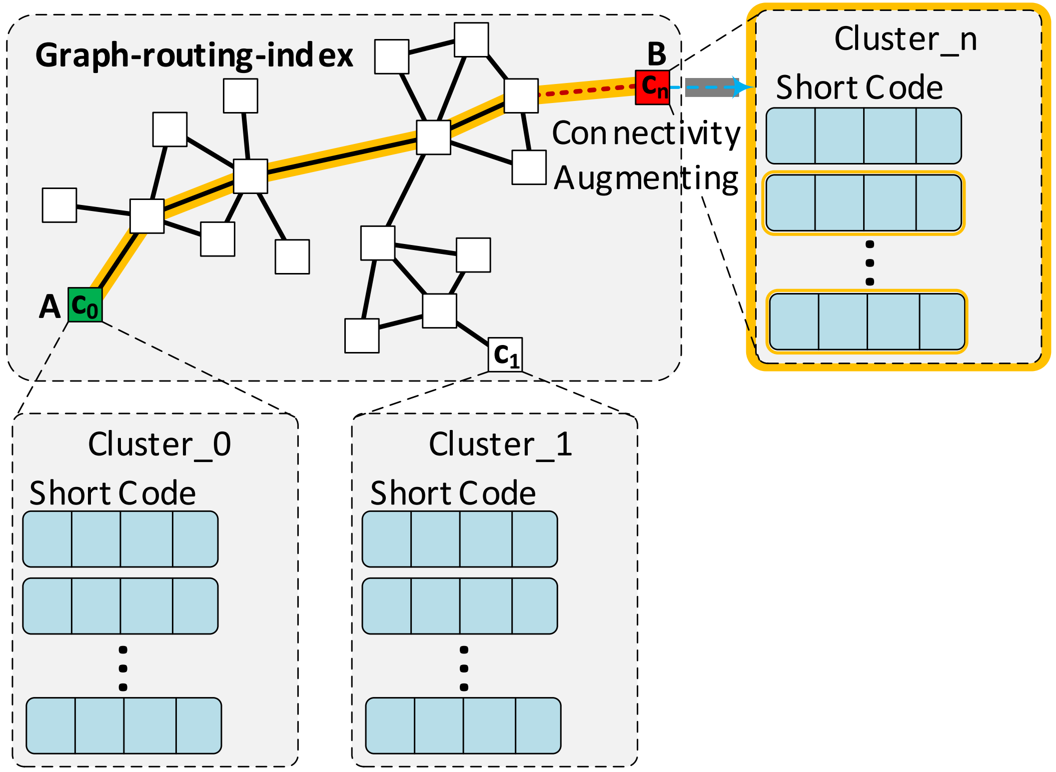
- **Challenges:** Memory capacity becomes a scalability bottleneck as the #vectors grows
  - E.g., HNSW takes 51G to index 80 millions of 128d feature vectors, which do not fit in 32G main memory



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

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

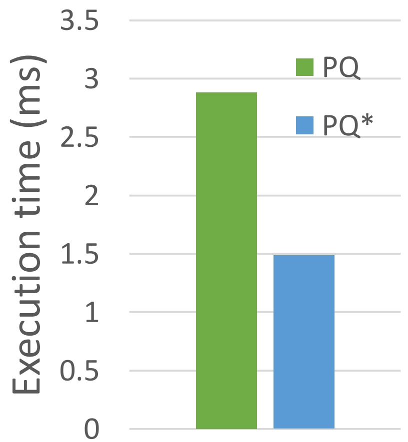
- **Challenge:** Distance estimation between the unquantized q and quantized short code can be slow

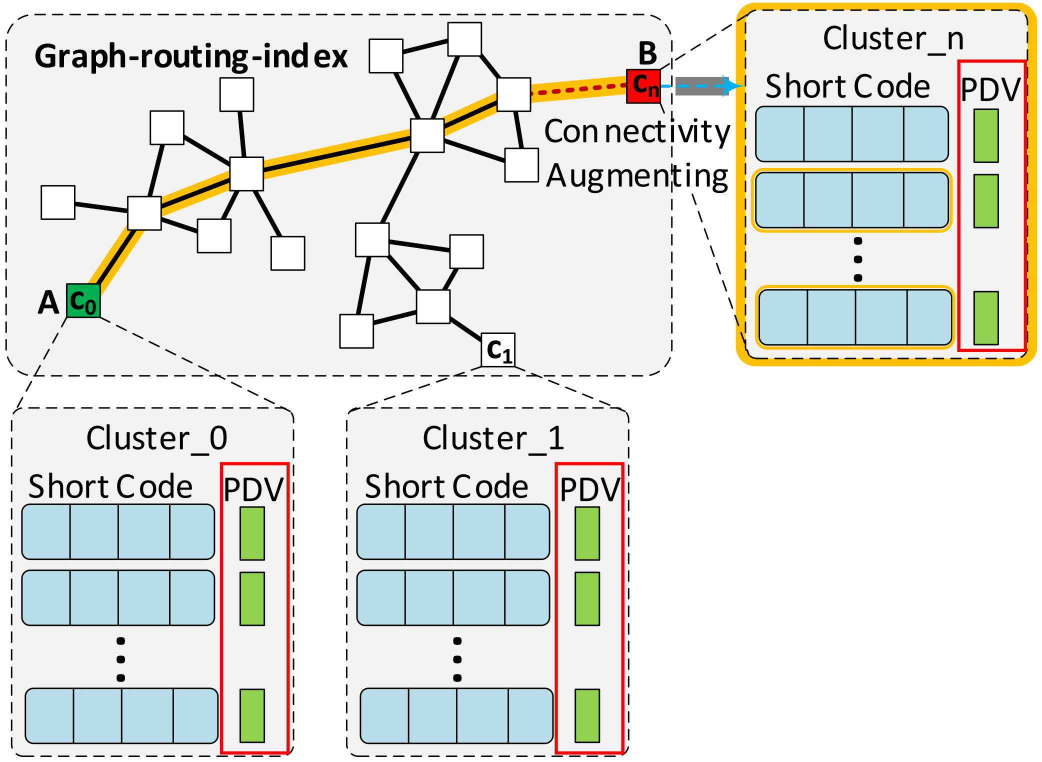
$$\underbrace{\|q-c\|^2}_{Term-A} + \underbrace{\sum_{m=1}^{M} \|c^m_{x^m}\|^2}_{Term-B} + \underbrace{2\sum_{m=1}^{M} \langle c^m, c^m_{x^m} \rangle}_{Term-C} - \underbrace{2\sum_{m=1}^{M} \langle q^m, c^m_{x^m} \rangle}_{Term-D}$$

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

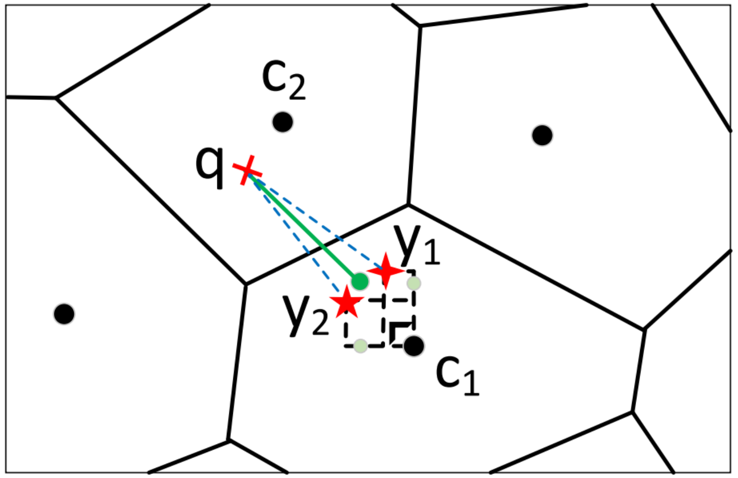




Pre-compute partial-distance value (PDV) offline and trade-off 1-float per vector for M lookups

Accuracy Boost

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

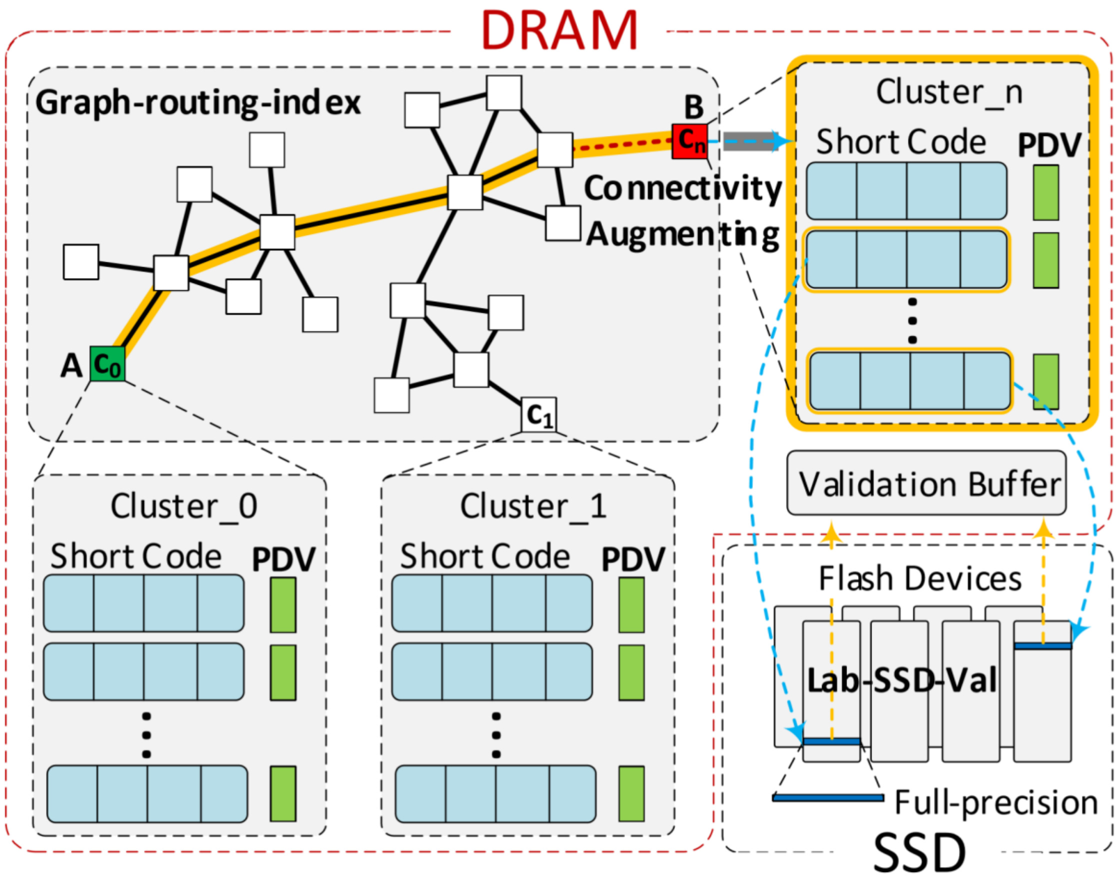


Vectors (y1, y2) quantized to the same short code (c1) have the same estimated distance to q

Accuracy Boost

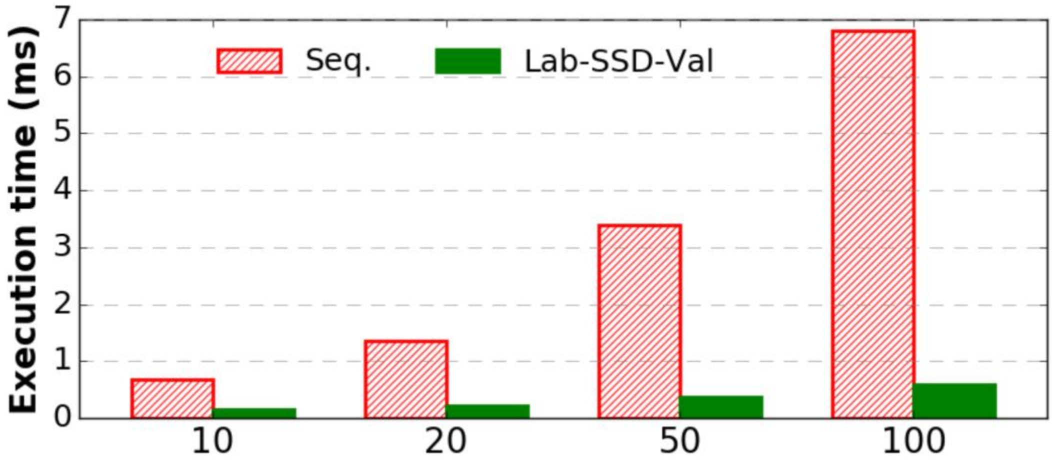
- **Solution:** Keep a copy of full-precision vectors on SSDs and validate a short candidate list from in-memory search

	DRAM	SSD
Capacity	Low	High
Cost	High	Low
Power consumption	High	Low
Scalability	GBs per DIMM	TBs per PCIe
Latency	Low	High



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 x Q

$$= \frac{\#Vectors}{Machine} \times \underline{Q}\text{uery processing rate}$$

Evaluation Metrics

- Recall
- Latency
- Memory cost
- *VQ* = V x Q

$$= \frac{\#Vectors}{Machine} \text{ x } \underline{\text{Q}}\text{uery processing rate}$$

$$\#\text{Machines} = \frac{\#Vectors\ X\ Q}{VQ}$$

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**

	NS	IVFPQ			GRIP			VQ
		Recall	Lat.	Mem.	Recall	Lat.	Mem.	Impr.
SIFT1M	1024	0.679	8.8	40	**0.904** $(NS:32)$	**0.6**	49	**12.0X**
	512	0.678	5.4	40	**0.989**	**1.8**	49	**2.4X**
	256	0.676	3.2	40	0.986	1.2	49	2.2X
	64	0.662	2.0	40	0.948	0.7	49	2.3X
Deep10M	1024	0.602	18.5	302	**0.906** $(NS:64)$	**1.2**	377	**12.3X**
	512	0.601	15.6	302	**0.975**	**5.4**	377	**2.3X**
	256	0.599	14.1	302	0.965	3.4	377	3.3X
	64	0.580	12.8	302	**0.906**	**2.0**	377	**5.1X**
SpaceV80M	1024	0.767	28.1	2552	**0.925** $(NS:32)$	**1.2**	4036	**14.8X**
	512	0.765	27.8	2552	**0.977**	**4.7**	4036	**3.7X**
	256	0.763	27.5	2552	0.971	2.7	4036	6.4X
	64	0.752	27.2	2552	**0.946**	**1.4**	4036	**12.3X**

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**

	HNSW				GRIP				VQ Improvement
	Recall	efSearch	Latency	Memory	NS	Recall	Latency	Memory	
SIFT1M	0.993	1280	2.3	588	512	**0.989**	1.8	**49**	**15.3X**
	0.973	320	0.6	588	128	0.976	0.8	49	9.0X
	0.947	160	0.3	588	64	0.948	0.7	49	5.1X
Deep10M	0.998	1280	3.1	4662	512	**0.994**	3.9	**377**	**9.8X**
	0.985	320	0.9	4662	256	0.983	2.6	377	4.3X
	0.969	160	0.4	4662	128	0.961	2.0	377	2.5X
SpaceV80M	0.972	2560	4.1	57554	512	**0.977**	4.7	**4036**	**12.4X**
	0.943	640	1.4	57554	128	0.961	1.8	4036	11.1X
	0.918	320	0.9	57554	64	0.946	1.4	4036	9.2X

Cost Comparison

	IVFPQ (Product quantization)	HNSW (Proximity graphs)	GRIP
Low search latency	✗	✓	✓
High accuracy	✗	✓	✓
Low memory cost	✓	✗	✓

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