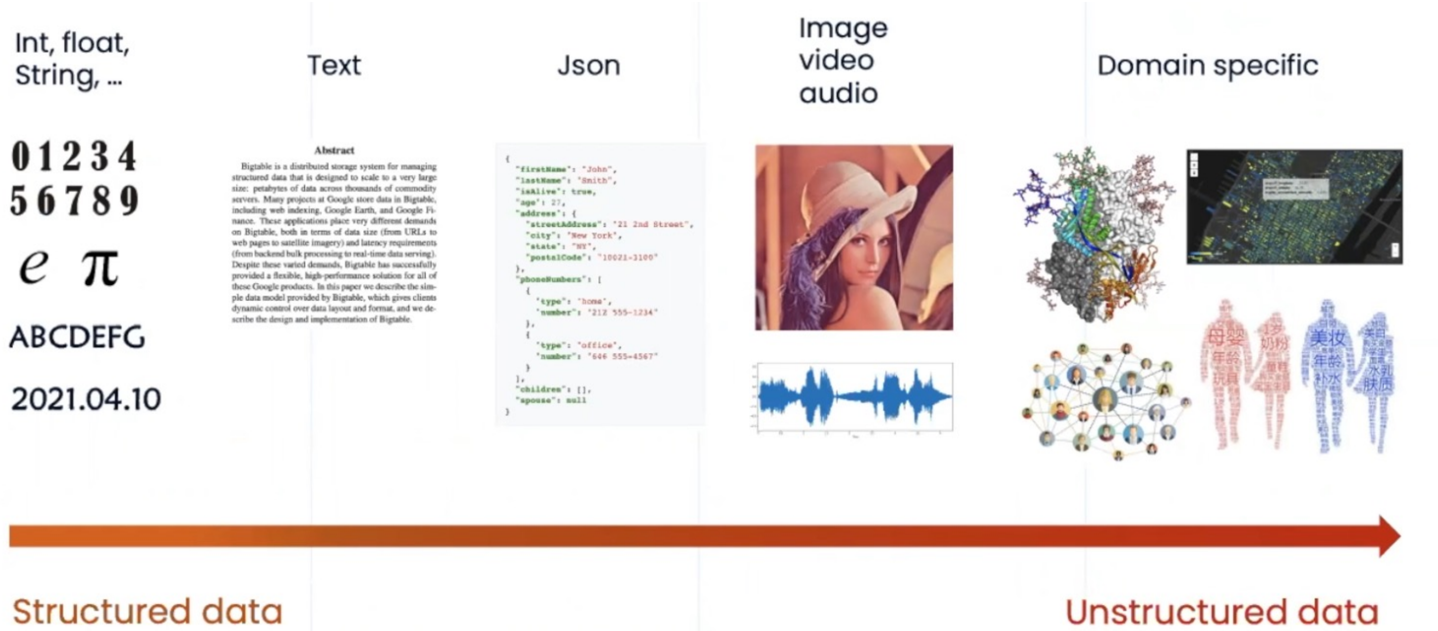
Milvus: A Purpose-Built Vector Data Management System

Jianguo Wang etc.

SIGMOD 2021

Background: Unstructured Data are Dominating



IDC: 80% data will be unstructured by 2025

Background: Vector Analysis Work Flow

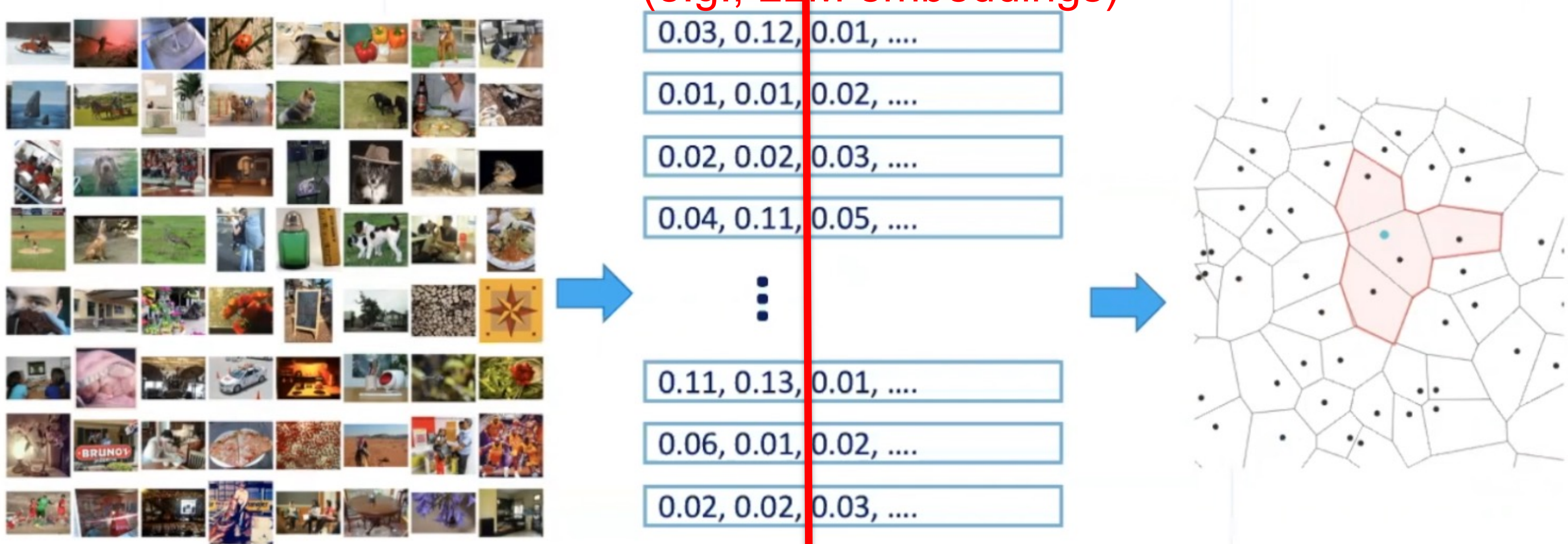
Machine learning (ML)

Vector Data Management System

Data

Vector
(e.g., LLM embeddings)

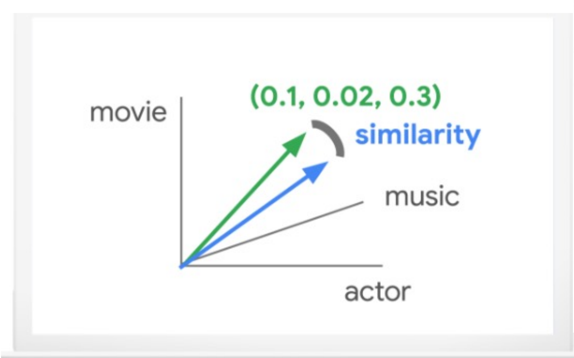
Analysis in vector space

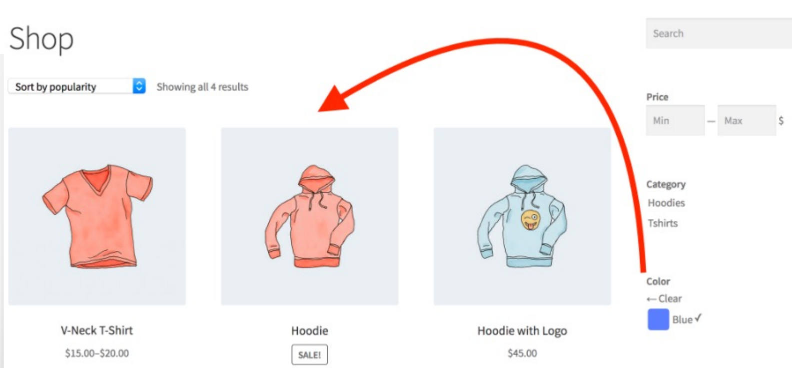


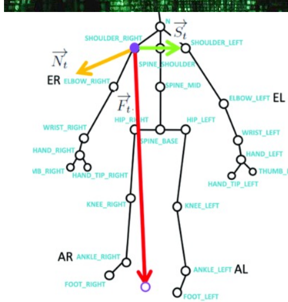
ML is soaring and bringing increasingly large volume of vectors, and how about the Vector Data Management System?

Requirements and Challenges for Scalable Vector Data Management System

- **Fast** query processing and **dynamic** vector manipulation
- **Advanced** query semantics:
  - Vector similarity search
  - Attribute filtering
  - Multi-vector search





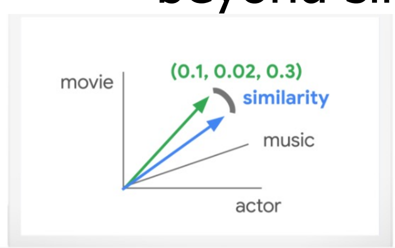


However, existing work fail
for poor performance
and limited functionalities...

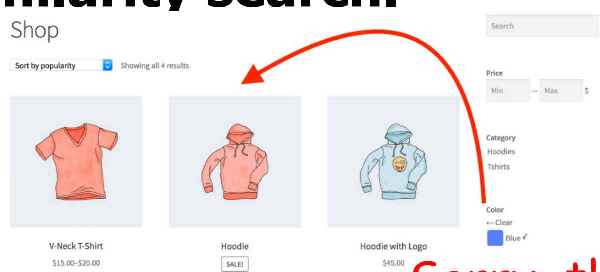
Limitations of Existing Work

	Billion-Scale Data	Dynamic Data	GPU	Attribute Filtering	Multi-Vector Query	Distributed System
Facebook Faiss [3, 35]	✓	✗	✓	✗	✗	✗
Microsoft SPTAG [14]	✓	✗	✗	✗	✗	✗
ElasticSearch [2]	✗	✓	✗	✓	✗	✓
Jingdong Vearch [4, 39]	✗	✓	✓	✓	✗	✓
Alibaba AnalyticDB-V [65]	✓	✓	✗	✓	✗	✓
Alibaba PASE (PostgreSQL) [68]	✗	✓	✗	✓	✗	✗
Milvus (this paper)	✓	✓	✓	✓	✓	✓

- Existing algorithms (e.g., [14]) don't support large data storage at scale or dynamic data with fast updates.
- Existing systems (e.g., [3]) simply extend relational DB and fail to fully optimize query processing on vector data
- Both lack the support for more advanced query processing beyond simple **vector similarity search.**

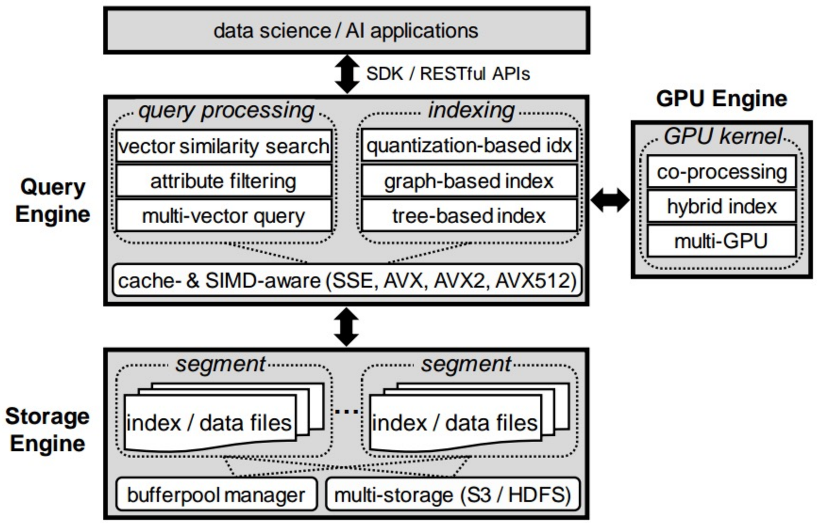


Yes, they can.



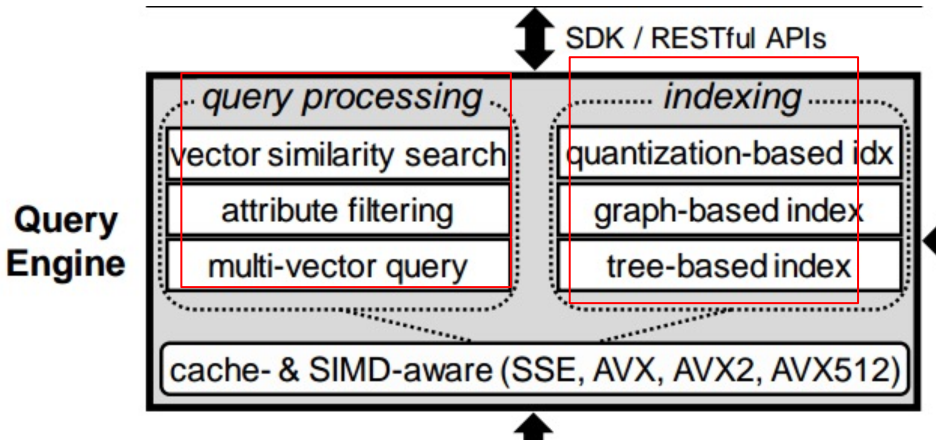
Sorry, they can not!!

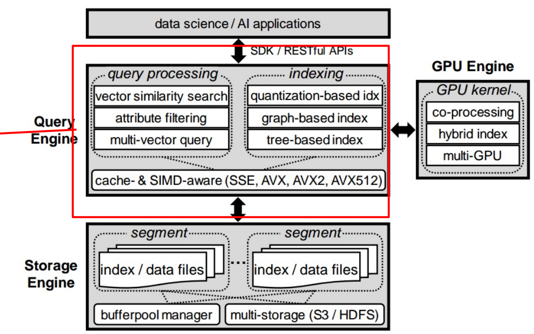
Milvus Overview



- All-in-one solution
  - Functional complete system + optimized algos
  - Hardware conscious design on both CPU and GPU
  - Easy SDK in Python/Java/Go/C++ and RESTful APIs
- Three major components
  - Query Engine
  - GPU Engine
  - Storage Engine

The Query Engine



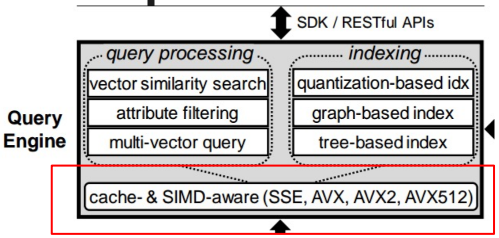


- Support vector(s) and vector(s)+non-vector attributes, therefore 3 primitive advanced queries*
- Cover common similarity and index functions
- Adopts LSM-tree [47] for efficient insertions and deletions
  - First in MemTable
  - Next flush into disk by rules (e.g., by size or time)

*Recall: Existing vectorDB only support similarity search

47: Chen Luo and Michael J. Carey. LSM-based Storage Techniques: A Survey. VLDBJ 2020

The Query Engine



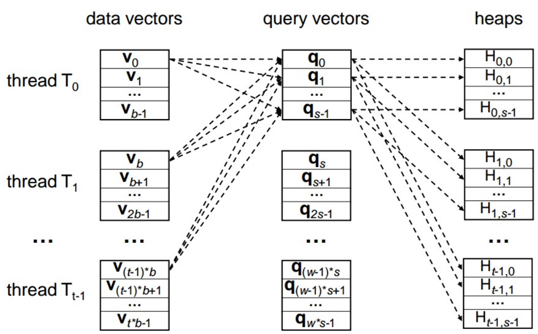
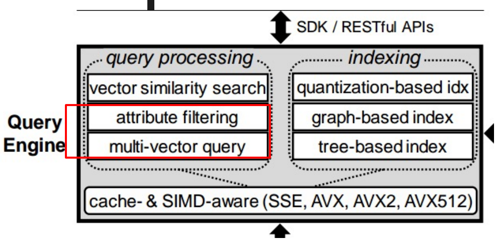


Figure 3: Cache-aware design in Milvus

- Cache-aware and SIMD optimizations
  - Cache-aware
    - Reuse accessed data for L3 opt.
      - Hence reduce L3 miss
    - Fine-grained parallelism of core-datablock assigns
      - Outperform OpenMP
  - SIMD
    - AVX512 support and automatic instruction choose in <u>runtime</u>

The Query Engine



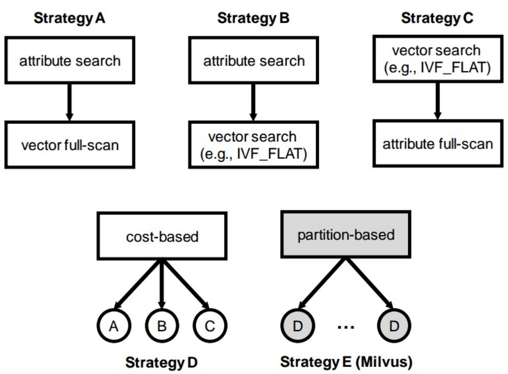
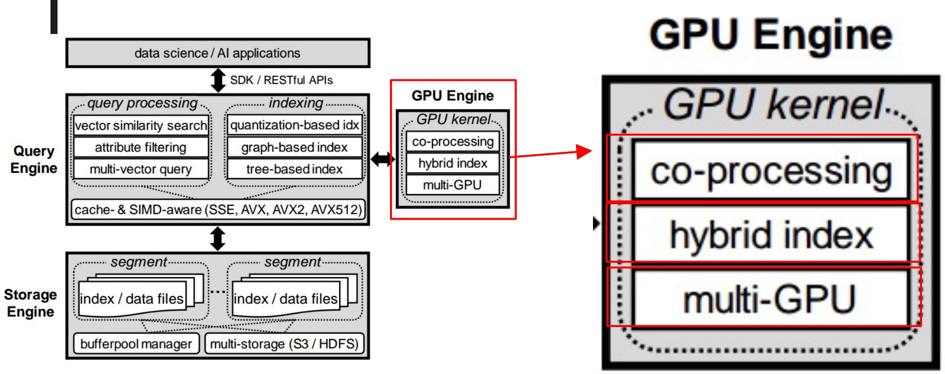


Figure 4: Different strategies for attribute filtering

- Further opt. on <u>attribute filtering and multi-vector query*</u>
  - Four adopted + one newly proposed attribute filtering
    - Partition-based is up to 13.7x faster than existing algos
  - Two new approaches instead of naïve solution for multi-vector query
    - i.e., vector fusion and iterative merging

*Recall: Not supported by existing vector database

The GPU Engine



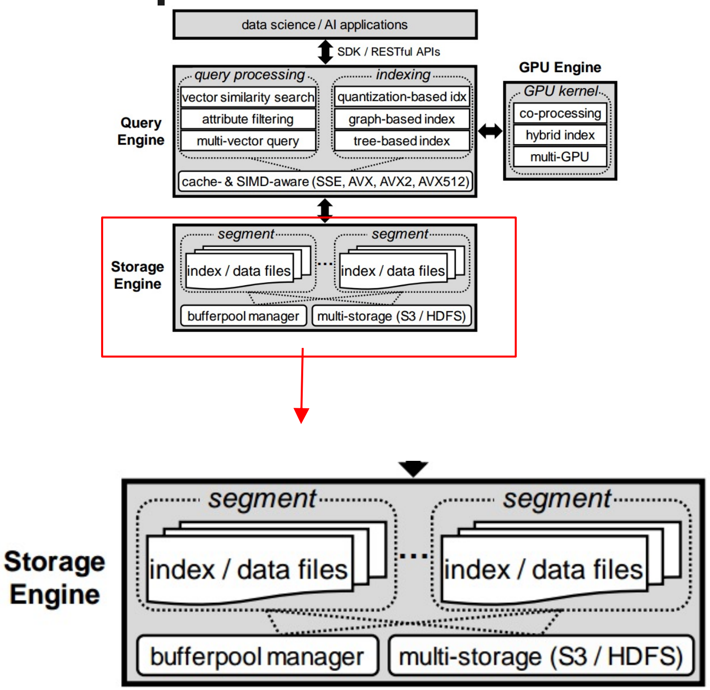
Algorithm 1: SQ8H

let n_q be the batch size;
if $n_q \geq$ *threshold* **then**
  run all the queries entirely in GPU (load multiple buckets to GPU memory on the fly);
else
  execute the step 1 of SQ8 in GPU: finding n_{probe} buckets;
  execute the step 2 of SQ8 in CPU: scanning every relevant bucket;

- Optimized SQ8 algo [3] for CPU-GPU co-processing
  - Communication opt. by multi-bucket copying
  - Computation opt. by fine-grained decomposition
- Optimized GPU kernel to supported larger query within GPU memory
  - <u>Progressively, round-by-round</u> flushing results, instead of loading the entire and flushing only once
- Multi-GPU support by adjusting device number in runtime

3: 2020. Facebook Faiss. https://github.com/facebookresearch/faiss

The Storage Engine



- Support various storage for different data structures and storage hardware
  - Vector storage (mem)
    - Continuous row for single vector
    - Columnar fashion for multiple vectors
  - Attribute storage (mem)
    - <key,value> pairs for each attribute column
    - Further optimize by building skip pointers[16]
  - Buffer pool to bridge mem and disk
  - Various file system support for in-disk and in-network

16: Benoit Dageville et al. The Snowflake Elastic Data Warehouse. SIGMOD 2016.

Implementation Optimizations

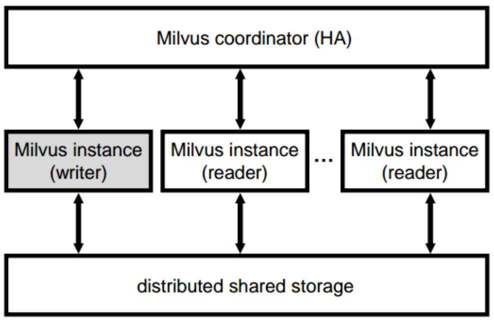
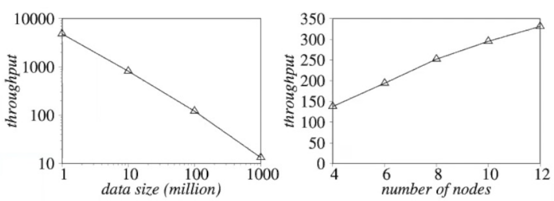


Figure 5: Milvus distributed system



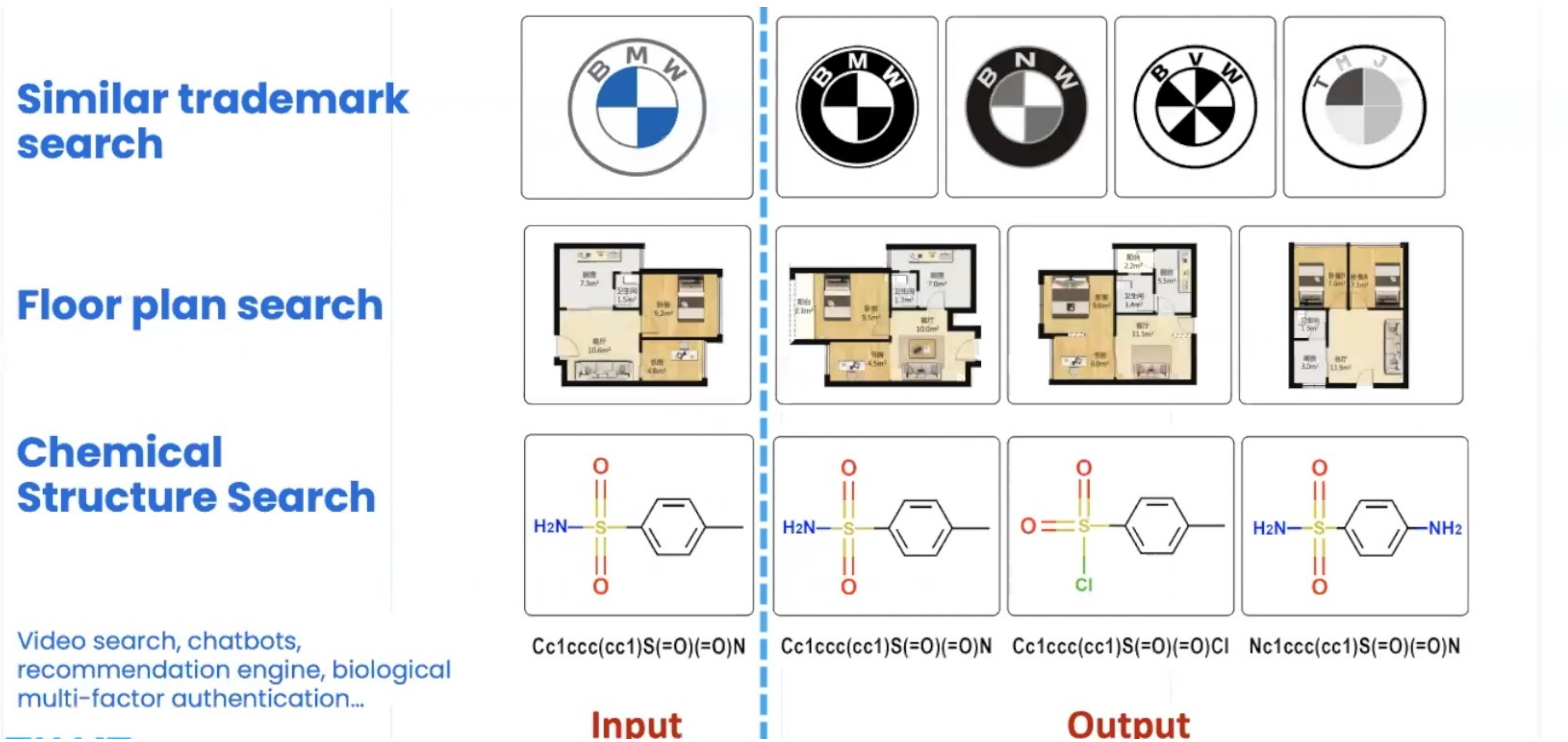
(a) Varying data size

(b) Varying #nodes

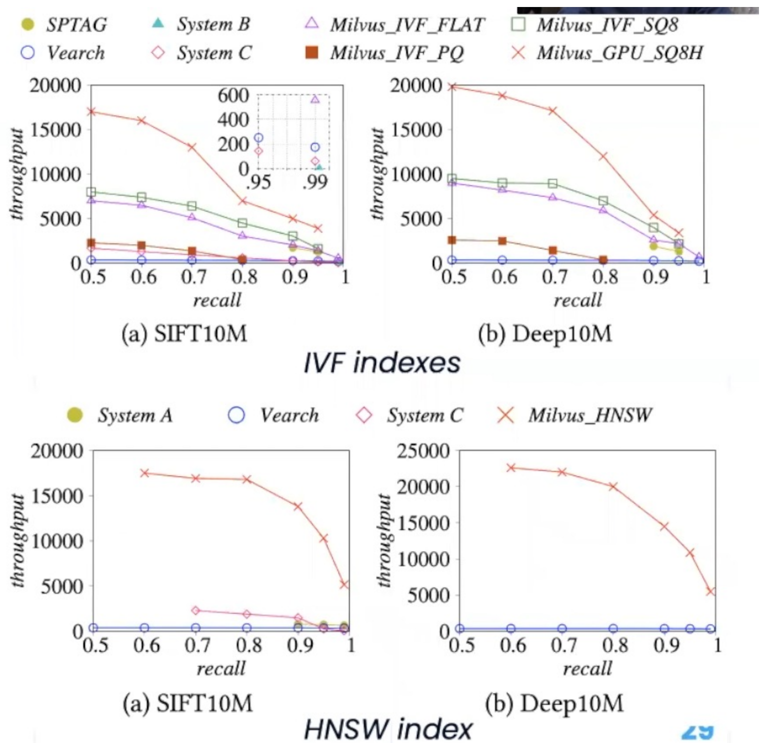
- Asynchronous processing
  - To minimize foreground processing (e.g., receiving request)
  - Write-ahead logs + operating at back gound
- Snapshot isolation
  - To make sure of the R/W consistency
  - Using LSM structure[47]
- Distributed system
  - For scalability and availability
  - Shared-storage and separated compute/storage
  - Stateless computing instances
    - Single writer multiple reader as Milvus is read-heavy

47: Chen Luo and Michael J. Carey. LSM-based Storage Techniques: A Survey. VLDBJ 2020

Application Examples



Outperform SOTA Significantly



- Platform: Alibaba cloud
- Dataset: SIFT1B and Deep1B
  - Both are 1 billion, 128-dimension
- Open source SOTA: SPTAG, Vearch
- Commercial system: System A, B, C

Conclusion and Future Work

- Milvus satisfies the requirements of versatile applications by heuristic optimization in system and algorithm

System:

- SIMD support
- GPU accelerations
- Snapshot isolation
- Cache-aware optimization
- LSM structure
- Distributed processing

Algorithm:

- Hybrid search algorithm
- Cost base optimization
- Vector fusion
- Iterative merging

- Future work can be following:
  - Cloud native architecture
  - FPGA acceleration

What Can We Learn and Do?

- This is an extremely complete system research.
  - Inclusively enough for wide community use

System:

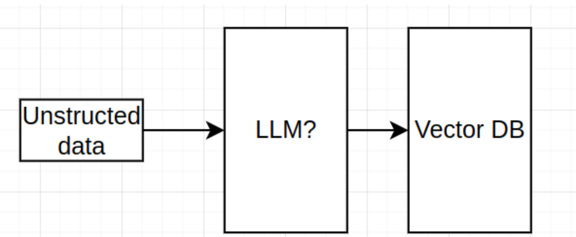
- SIMD support
- GPU accelerations
- Snapshot isolation
- Cache-aware optimization
- LSM structure
- Distributed processing

Algorithm:

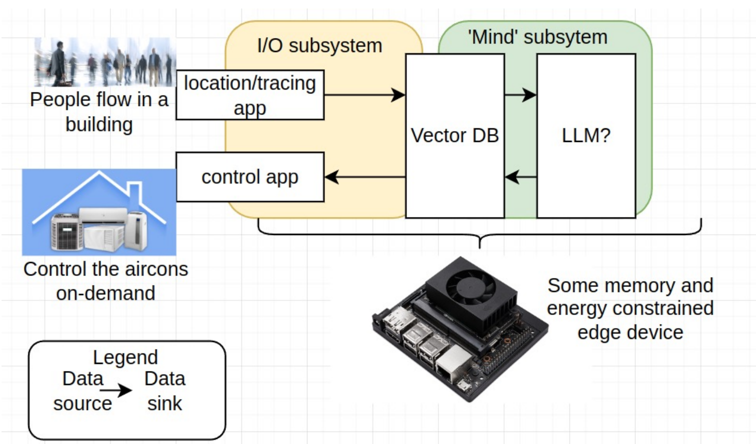
- Hybrid search algorithm
- Cost base optimization
- Vector fusion
- Iterative merging

But it is not a good idea to apply Milvus on Edges...

What if On Edge?



Their proposed data flow at the cloud



One possible edge application

- Our toy edge application
  - Continuously tracing people flow and control the aircons on demand
  - VectorDB bridges two subsystems on a device
- Problem 1: The indexing and storage overhead is still too heavy for constrained edges
  - Is there any cure by using approximate computation, or even new indexing ways?
- Problem 2: Two subsystems may raise high-speed and concurrent write (streams)
  - Will stateless, single writer still work well?