



WELCOME

Embedding The Future

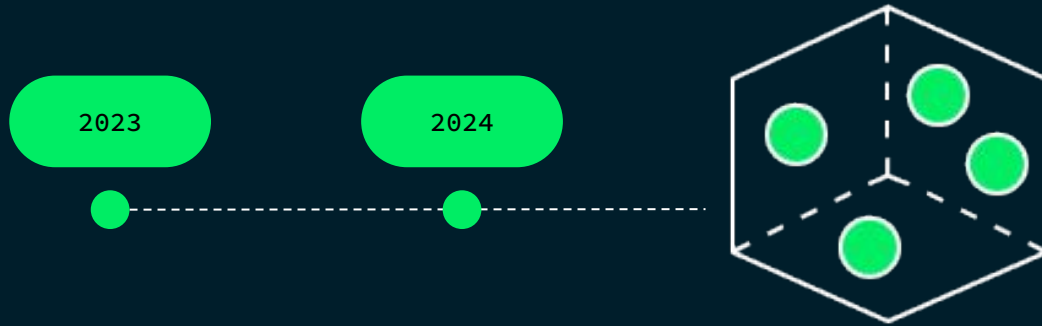
How Modern Databases are eating Search



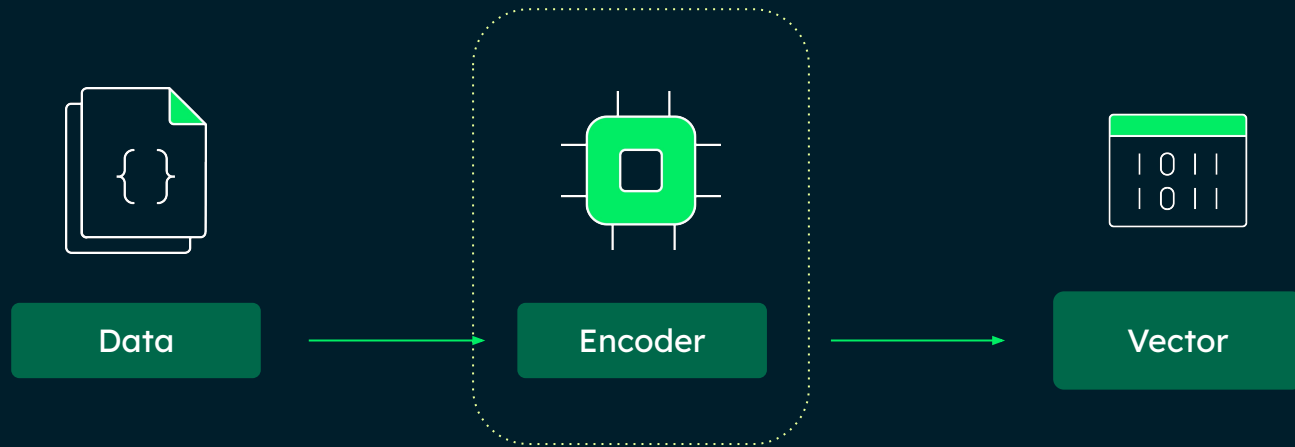
Philip Eschenbacher

Advisory Solutions Architect
Alps Region

Two years of Atlas Vector Search

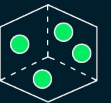


Vectors are produced by sending data through an encoding model



“a quick brown fox”

[0.743, 0.720, -0.325,
0.195, 0.835, -0.945,]



Vectors in Document



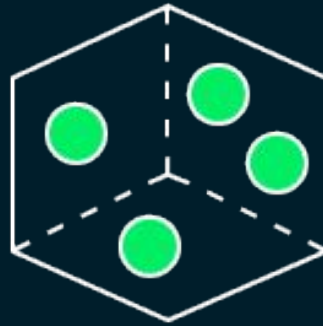
```
_id: ObjectId('62f13a3f7aecb216')
symbol: "ABMD"
quarter: 4
year: 2021
date: 2021-04-29T20:10:40.000+00:00
content: "Operator: Ladies and
gentlemen, thank you for standing by,
and welcome..."
```

```
content_embeddings: Array
  0.03898080065846443
 -0.05879044905304909
  0.04323238879442215
 -0.021337900310754776
 -0.036346953362226486
  0.028689613565802574
 -0.03514527902007103
 -0.07414846867322922
 -0.00993054173886776
  0.007234036456793547
 -0.03197460621595383
```

Embeddings are
stored as an
array of floats

More dimensions
= more floating
points

The next two years



Vision Statement



To meet the needs of next generation workloads, we must provide a vector search capability that is seamless to the rest of MongoDB.

Vision Statement



①

To meet the needs of **next generation workloads**, we must provide a vector search capability that is seamless to the rest of MongoDB.

Vision Statement



①

To meet the needs of **next generation workloads**, we must provide a **vector search capability** that is seamless to the rest of MongoDB.

②



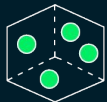
Vision Statement



①

To meet the needs of **next generation workloads**, we must provide a **vector search capability** that is seamless to the **rest of MongoDB**.

②



③



Next Generation Workloads



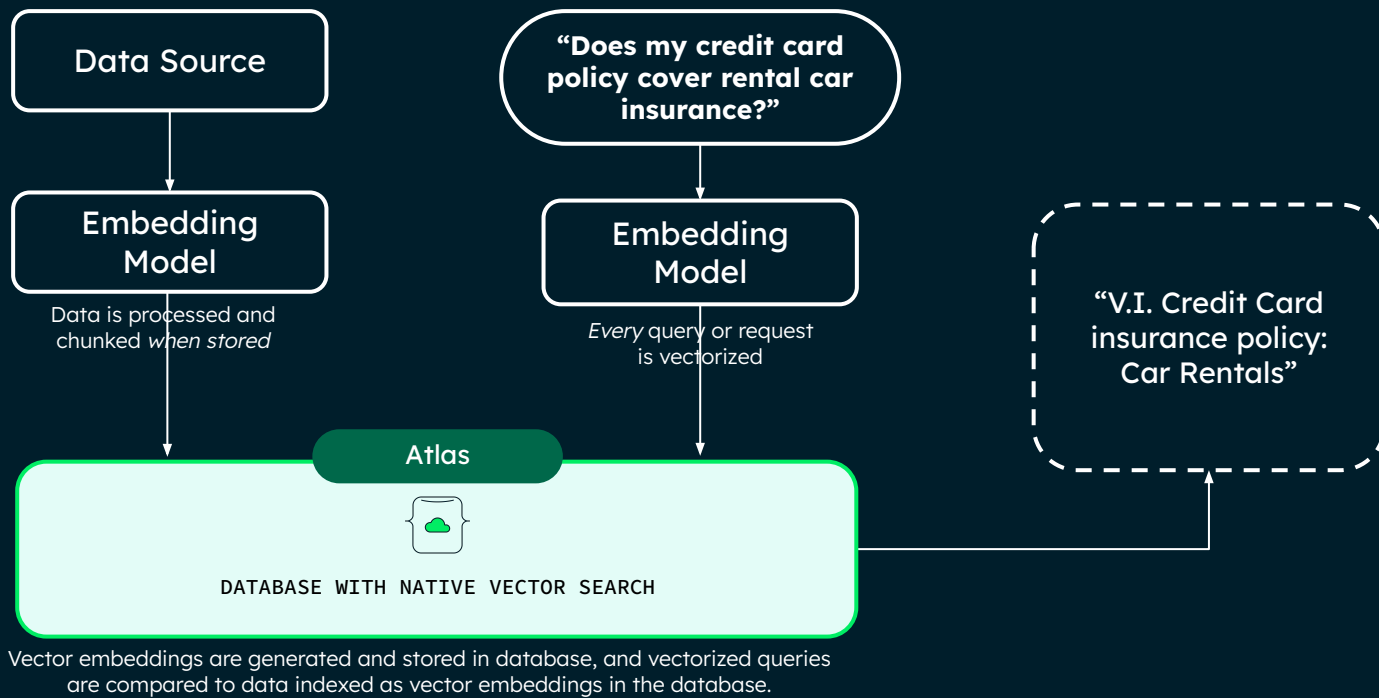
Learning from our Customers



What are our customers doing?



Semantic Search



Retrieval Augmented Generation (RAG)

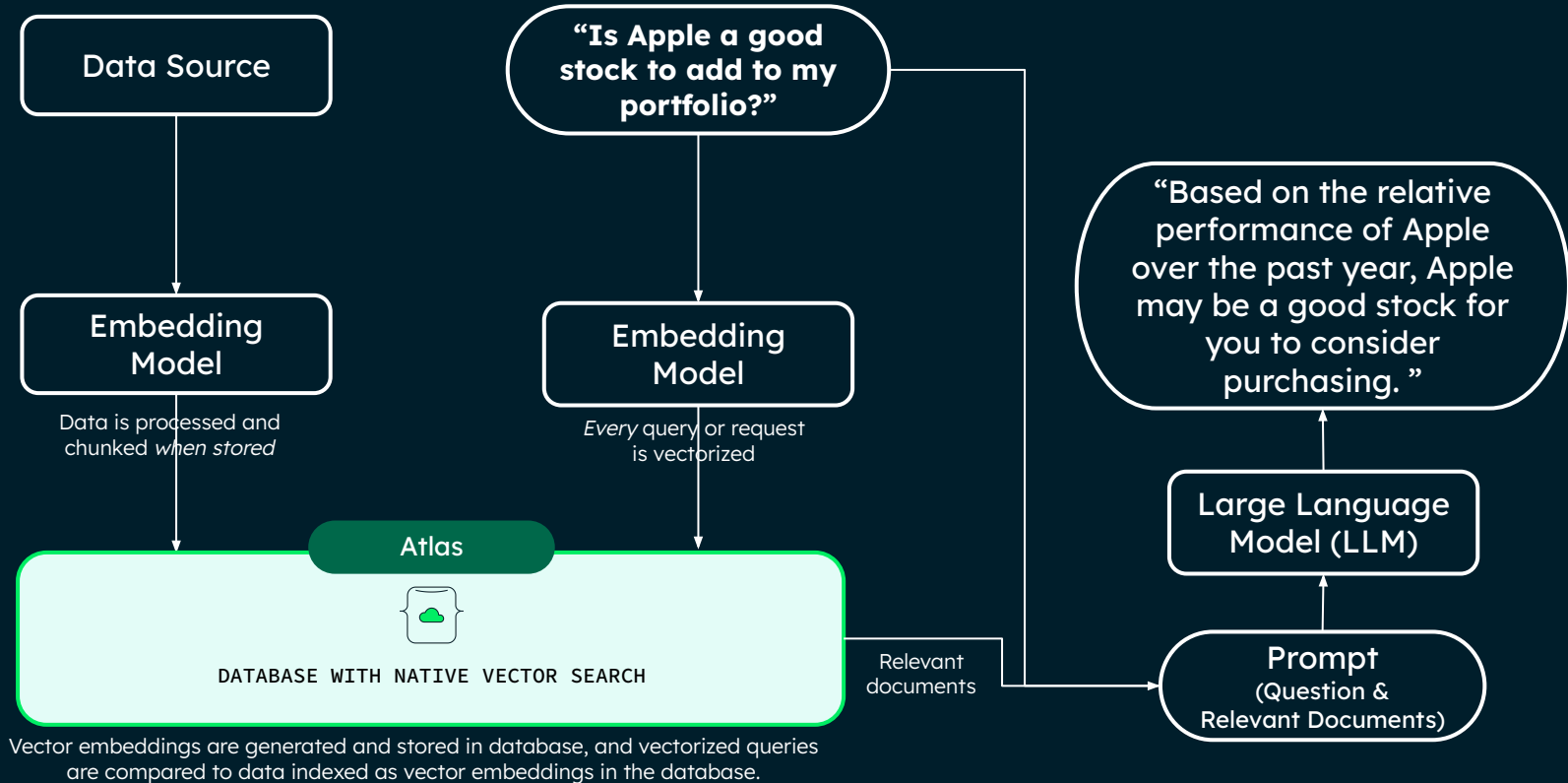


Image Similarity Search



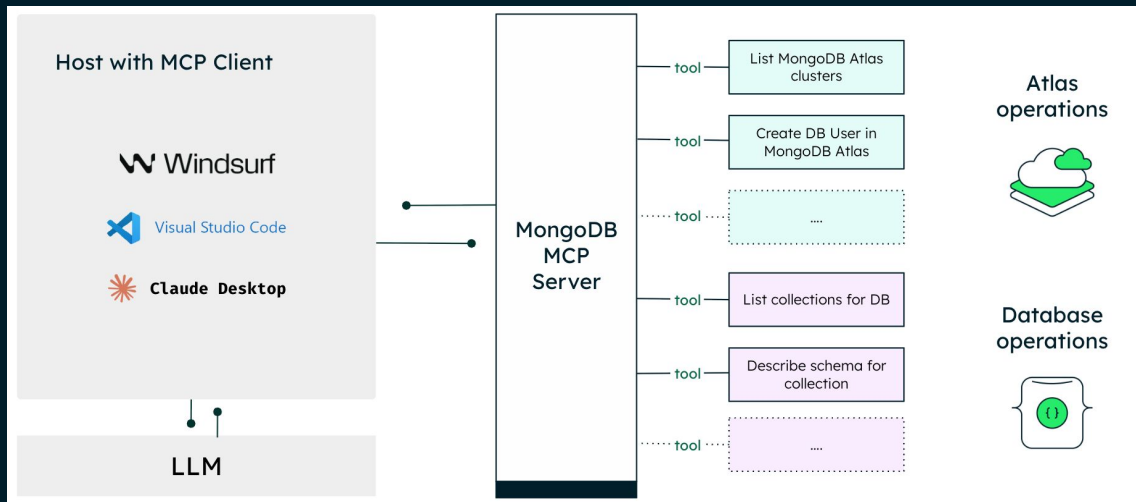
Choose similar images:



KYC + Fraud Detection



Accessing Databases via LLM



[Source: MongoDB](#)

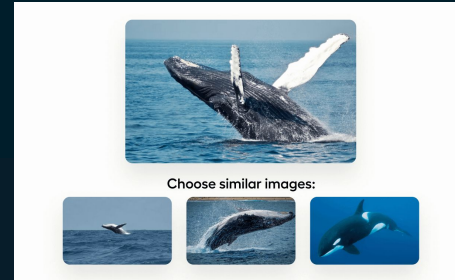
Change is Good



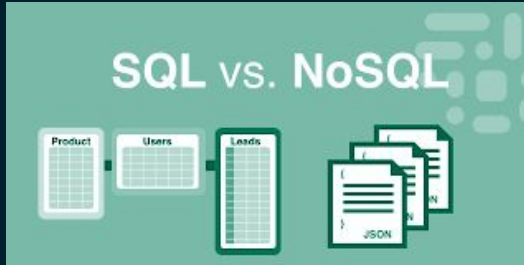
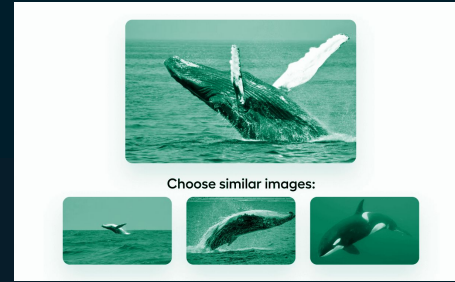
Change is Good



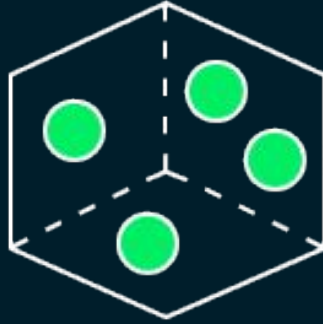
Change is Good



MongoDB is Built for Change



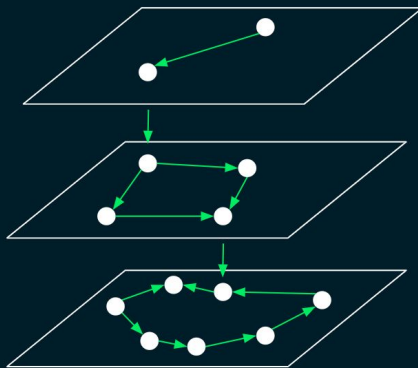
Vector Search Capability



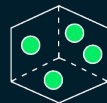
Vector Search Index Structure



Approximate Nearest Neighbor (ANN)
via HNSW for Fast Retrieval



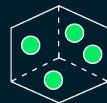
HNSW = Hierarchical Navigable Small Worlds



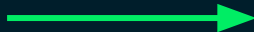
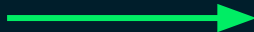
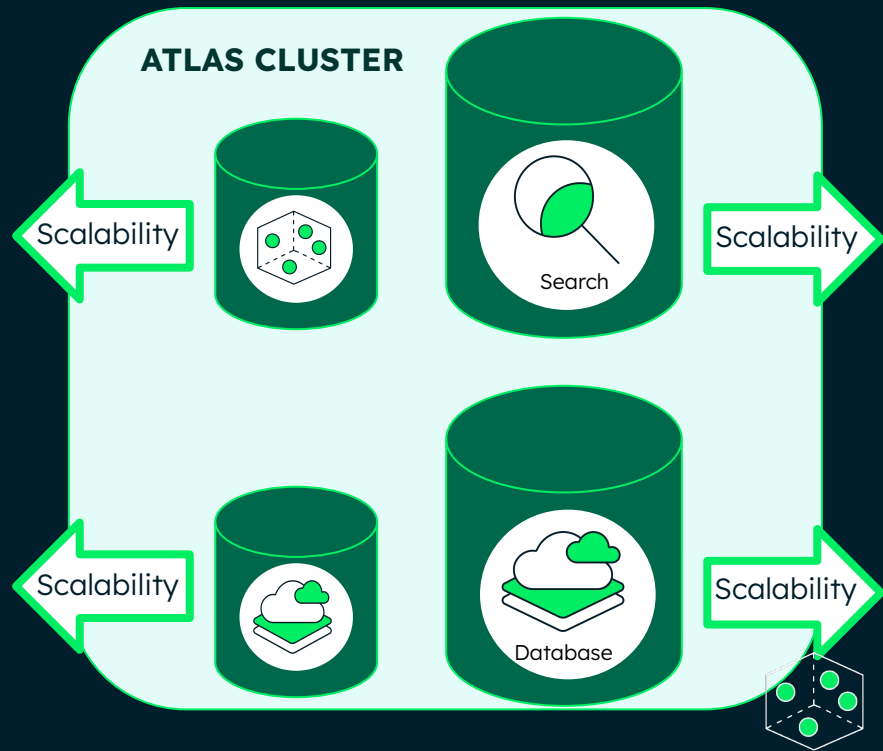
Key Scalability Features



1. Dedicated Search Nodes for high availability
2. Dedicated BSON type
3. Indexing quantized vectors



Dedicated Search Nodes



= Workload size



= Resources required



“

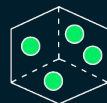
Upgrading the search node cluster had a marked impact on the query time; Vector search latency is 150-300 ms after the node upgrade - a 99% reduction in query time and a 90x performance improvement.



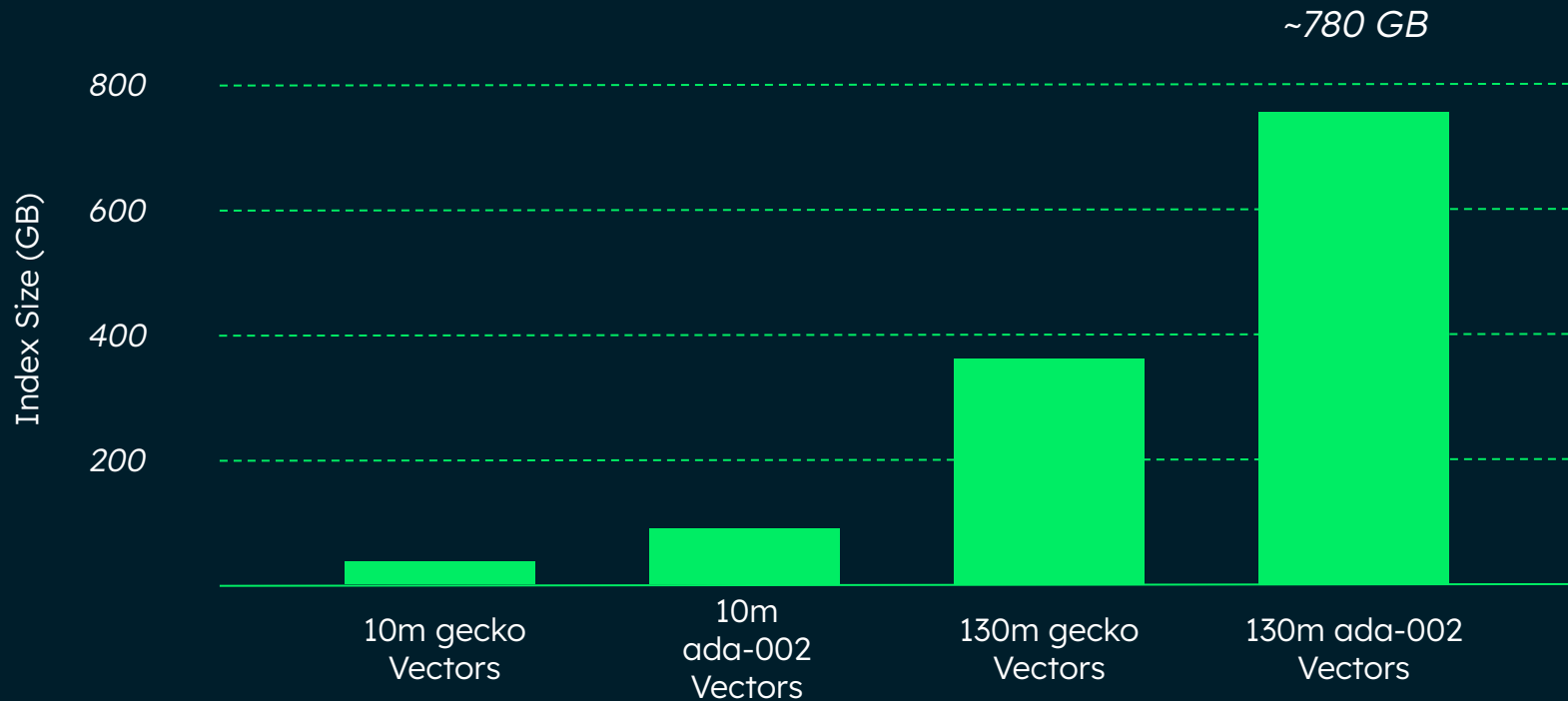
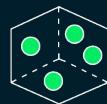
Phillip Kinney | Product Engineer | Fizz

Indexing Float Vectors

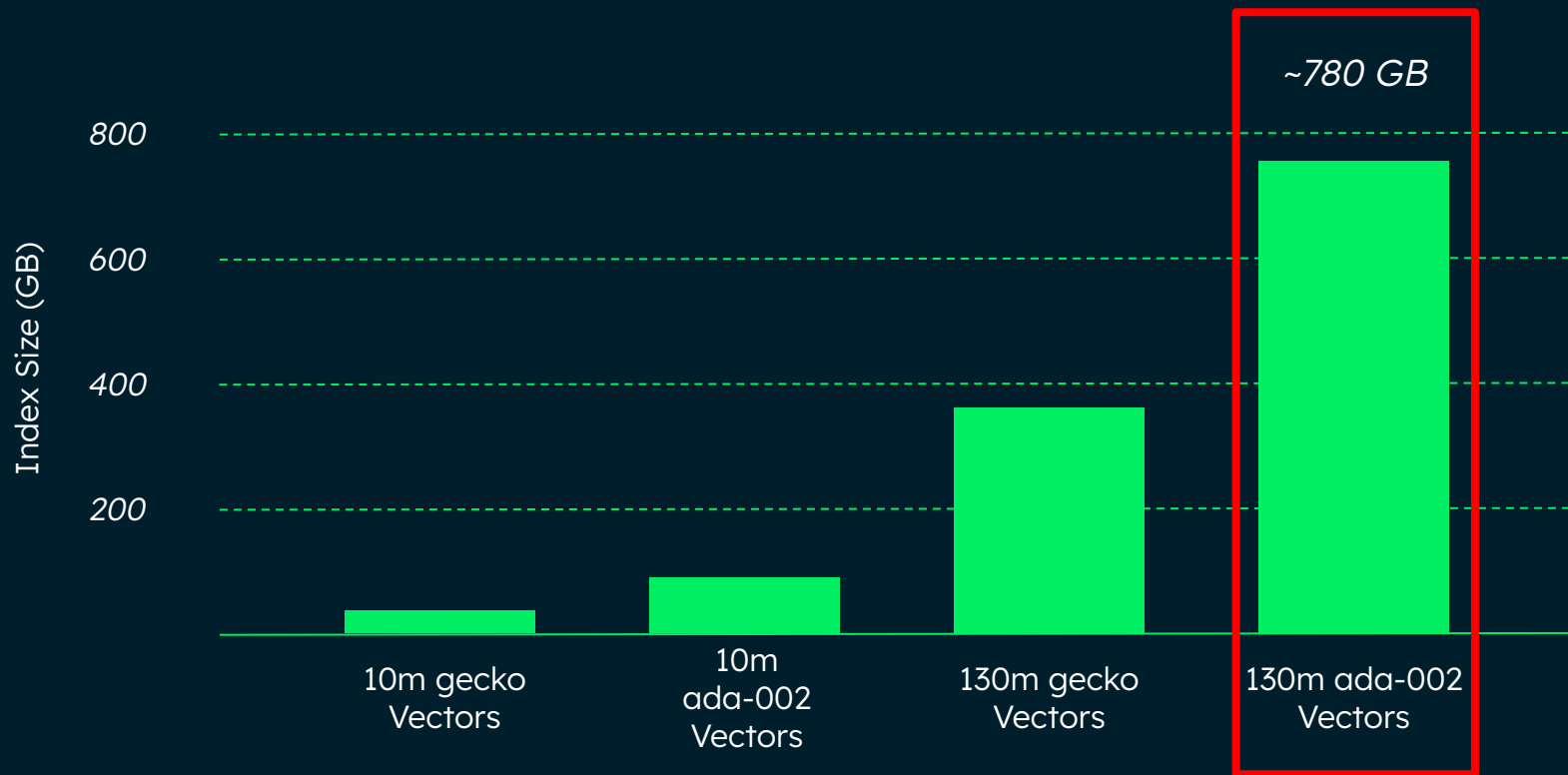
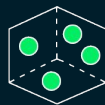
Embedding Model	Vector Dimension	RAM Required
OpenAI <code>text-embedding-ada-002</code>	1536	6kb
Google <code>text-embedding-gecko</code>	768	3kb



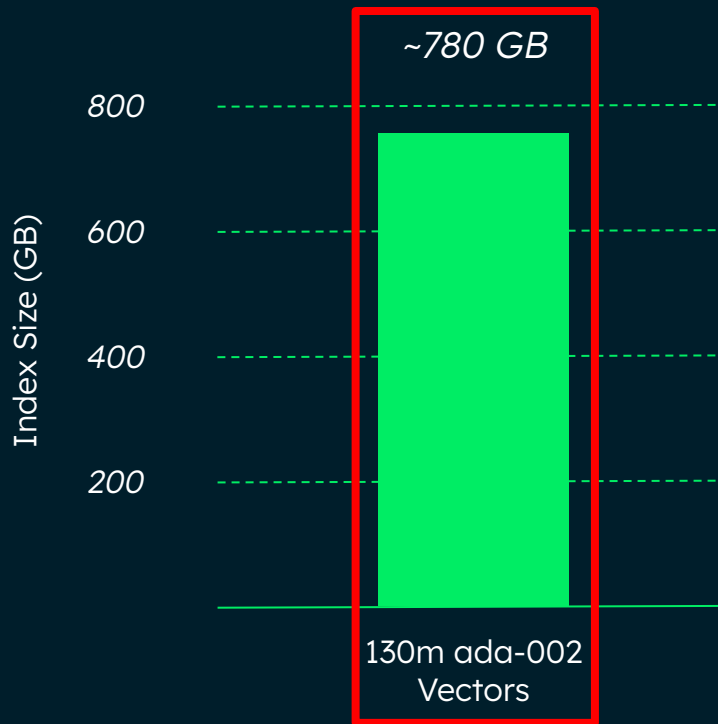
Indexing Float Vectors



Indexing Float Vectors



Indexing Float Vectors



Cluster Tier

M30 (8 GB RAM, 345 GB Storage)
Search S110 (512 GB RAM, 3200 GB Storage)
3,000 IOPS, Encrypted, Auto-expand Storage

Base Tier **Search Tier** NEW

Hourly price is for 1 Search Node and may change depending on the number of Search Nodes configured.

Low-CPU **High-CPU**

Tier	RAM	Storage	vCPU	Price
\$30	8 GB	50 GB	1 vCPU	from \$0.11 /hr
\$40	16 GB	100 GB	2 vCPUs	from \$0.21 /hr
\$50	32 GB	200 GB	4 vCPUs	from \$0.43 /hr
\$60	64 GB	380 GB	8 vCPUs	from \$0.89 /hr
\$80	128 GB	760 GB	16 vCPUs	from \$1.68 /hr
\$90	256 GB	1600 GB	32 vCPUs	from \$3.12 /hr
\$100	384 GB	2400 GB	48 vCPUs	from \$4.40 /hr
☒ \$110	512 GB	3200 GB	64 vCPUs	from \$5.87 /hr

Search Optimized Disk 860000 Read IOPS
360000 Write IOPS

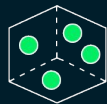
Additional Info 25 Gigabit network performance
[Learn more](#)

\$12.46/hour Pay-as-you-go! You will be billed hourly and can terminate your cluster anytime. Excludes variable data transfer, backup, and taxes. [Cancel](#) [Save Draft](#) [Review Changes](#)

Quantization solves for this!



```
{  
  "fields": [{  
    "type": "vector",  
    "path": "plot_embedding",  
    "numDimensions": 1536,  
    "similarity": "cosine",  
    "quantization": "none" | "scalar" | "binary"  
  }]  
}
```



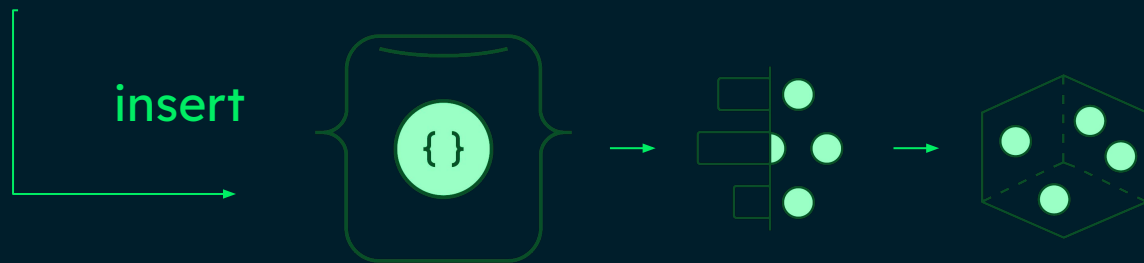
Scalar Quantization



[0.743, 0.720, -0.325, 0.195, 0.835, -0.945, ...]

[0.322, -0.065, 0.671, 0.234, 0.055, -0.321, ...]

[-0.189, 0.243, 0.098, -0.354, 0.119, 0.078, ...]



binData([0.743, 0.720, -0.325, ...])

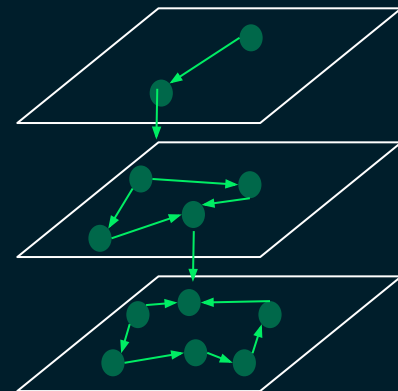
...

Index

[23, 6, 150, 8, 71, ...]

[91, 2, 48, 201, 1, ...]

[223, 8, 12, 84, 7, ...]



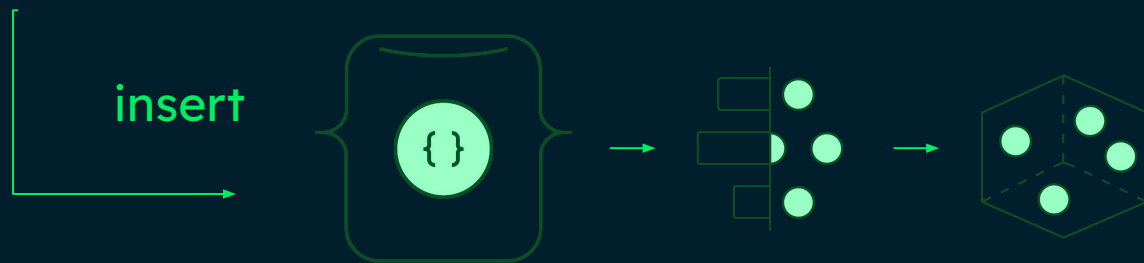
Binary Quantization



[0.743, 0.720, -0.325, 0.195, 0.835, -0.945, ...]

[0.322, -0.065, 0.671, 0.234, 0.055, -0.321, ...]

[-0.189, 0.243, 0.098, -0.354, 0.119, 0.078, ...]



binData([0.743, 0.720, -0.325, ...])

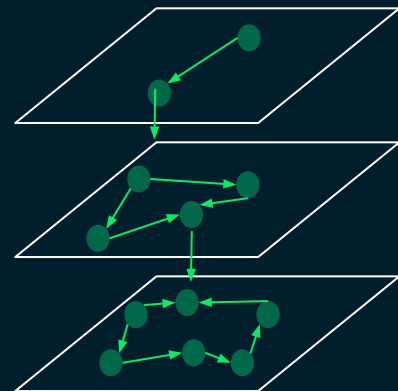
...

Index

[1, 1, 0, 1, 1, 0, ...]

[1, 0, 1, 1, 0, 1, ...]

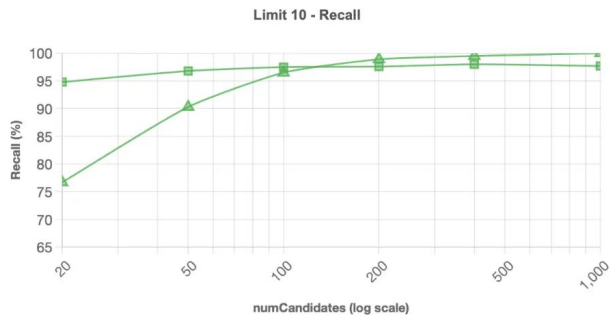
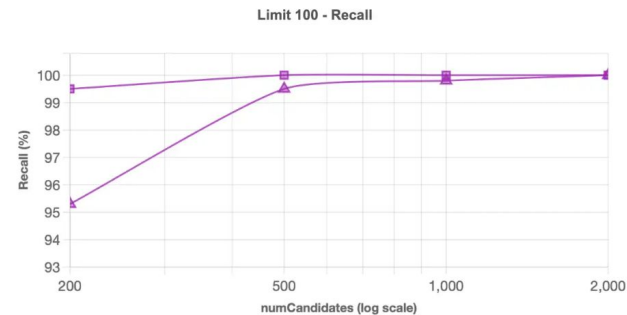
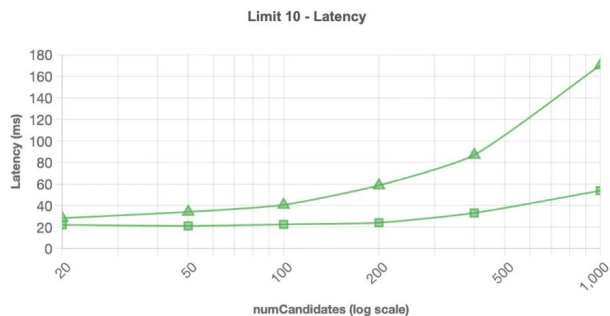
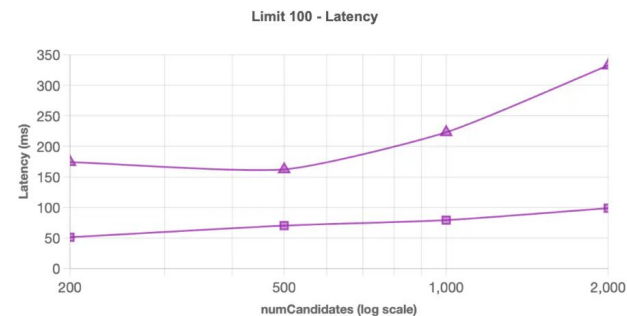
[0, 0, 0, 0, 0, 1, ...]



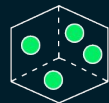
Binary vs Scalar Quantization Performance

15.3M Vector Dataset (2048D) • Recall & Latency Analysis

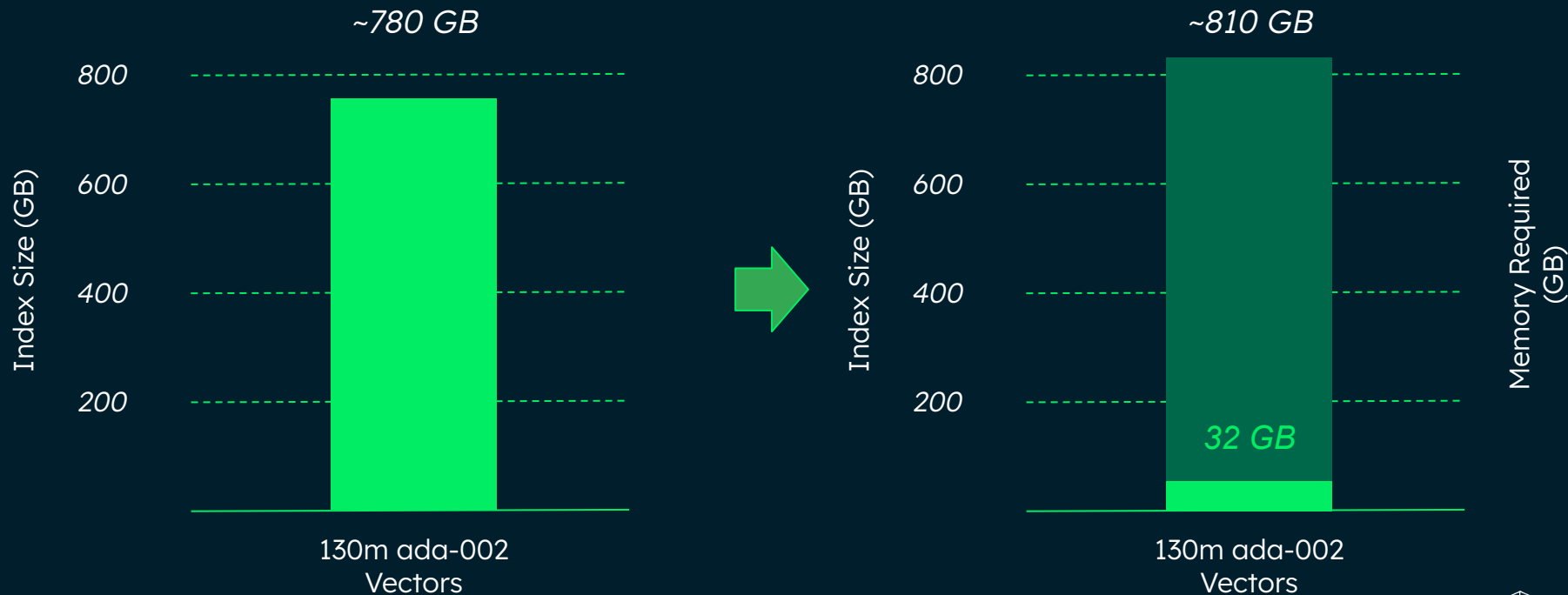
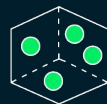
Limit 10 Limit 100 Quantization Type: ▲ Binary | ■ Scalar



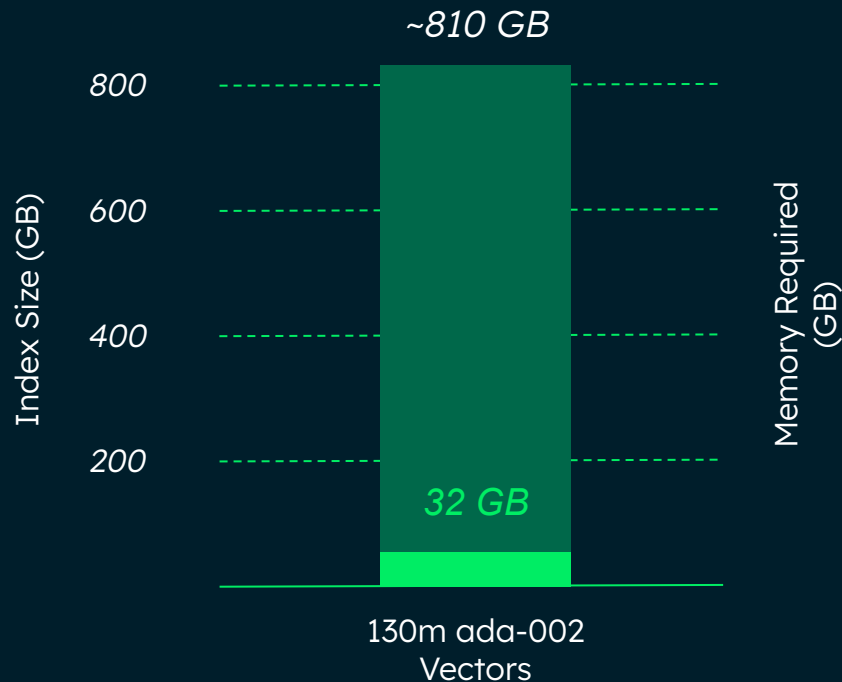
Source: MongoDB Vector Benchmark



Indexing Quantized Vectors



Indexing Quantized Vectors



Cluster Tier

M30 (8 GB RAM, 345 GB Storage)
Search S60 (64 GB RAM, 1600 GB Storage)
3,000 IOPS, Encrypted, Auto-expand Storage

Base Tier Search Tier **NEW**

Hourly price is for 1 Search Node and may change depending on the number of Search Nodes configured.

Low-CPU High-CPU

Tier	RAM	Storage	vCPU	Price
\$20	4 GB	100 GB	2 vCPUs	from \$0.12 /hr
\$30	8 GB	200 GB	4 vCPUs	from \$0.24 /hr
\$40	16 GB	380 GB	8 vCPUs	from \$0.48 /hr
\$50	32 GB	760 GB	16 vCPUs	from \$0.99 /hr
☒ \$60	64 GB	1600 GB	32 vCPUs	from \$1.77 /hr
Search Optimized Disk 430000 Read IOPS 180000 Write IOPS				
Additional Info 12 Gigabit network performance Learn more				
\$70	96 GB	2400 GB	48 vCPUs	from \$2.50 /hr
\$80	128 GB	3200 GB	64 vCPUs	from \$3.26 /hr

\$4.27/hour Pay-as-you-go! You will be billed hourly and can terminate your cluster anytime. Excludes variable data transfer, backup, and taxes.

[Cancel](#) [Save Draft](#) [Review Changes](#)

Storage Optimized Search Nodes



Cluster Tier

M30 (8 GB RAM, 345 GB Storage)
Search S60 (64 GB RAM, 1600 GB Storage)
3,000 IOPS, Encrypted, Auto-expand Storage

Base Tier

Search Tier **NEW**

Hourly price is for 1 Search Node and may change depending on the number of Search Nodes configured.

Low-CPU

High-CPU

Tier	RAM	Storage	vCPU	Price
\$20	4 GB	100 GB	2 vCPUs	from \$0.12 /hr
\$30	8 GB	200 GB	4 vCPUs	from \$0.24 /hr
\$40	16 GB	380 GB	8 vCPUs	from \$0.48 /hr
\$50	32 GB	760 GB	16 vCPUs	from \$0.99 /hr
☒ \$60	64 GB	1600 GB	32 vCPUs	from \$1.77 /hr
Search Optimized Disk	430000 Read IOPS 180000 Write IOPS			
Additional Info	12 Gigabit network performance Learn more			
\$70	96 GB	2400 GB	48 vCPUs	from \$2.50 /hr
\$80	128 GB	3200 GB	64 vCPUs	from \$3.26 /hr

\$4.27/hour

Pay-as-you-go! You will be billed hourly and can terminate your cluster anytime. Excludes variable [data transfer](#), [backup](#), and taxes.

Cancel

Save Draft

Review Changes



Cluster Tier

M30 (8 GB RAM, 345 GB Storage)
Search S60 (64 GB RAM, 1500 GB Storage)
3,000 IOPS, Encrypted, Auto-expand Storage

Base Tier

Search Tier **NEW**

Hourly price is for 1 Search Node and may change depending on the number of Search Nodes configured.

Low-CPU

High-CPU

Storage-Optimized

Tier	RAM	Storage	vCPU	Price
\$40	16 GB	375 GB	2 vCPUs	from \$0.25 /hr
\$50	32 GB	750 GB	4 vCPUs	from \$0.52 /hr
☒ \$60	64 GB	1500 GB	8 vCPUs	from \$0.94 /hr
Search Optimized Disk	125000 Read IOPS 100000 Write IOPS			
Additional Info	Up to 12 Gigabit network performance Learn more			
\$80	128 GB	3000 GB	16 vCPUs	from \$1.89 /hr
\$90	256 GB	6000 GB	32 vCPUs	from \$3.76 /hr

\$2.62/hour

Pay-as-you-go! You will be billed hourly and can terminate your cluster anytime. Excludes variable [data transfer](#), [backup](#), and taxes.

Cancel

Save Draft

Review Changes

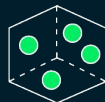
Additional Settings

MonaoDB 6.0. Backup

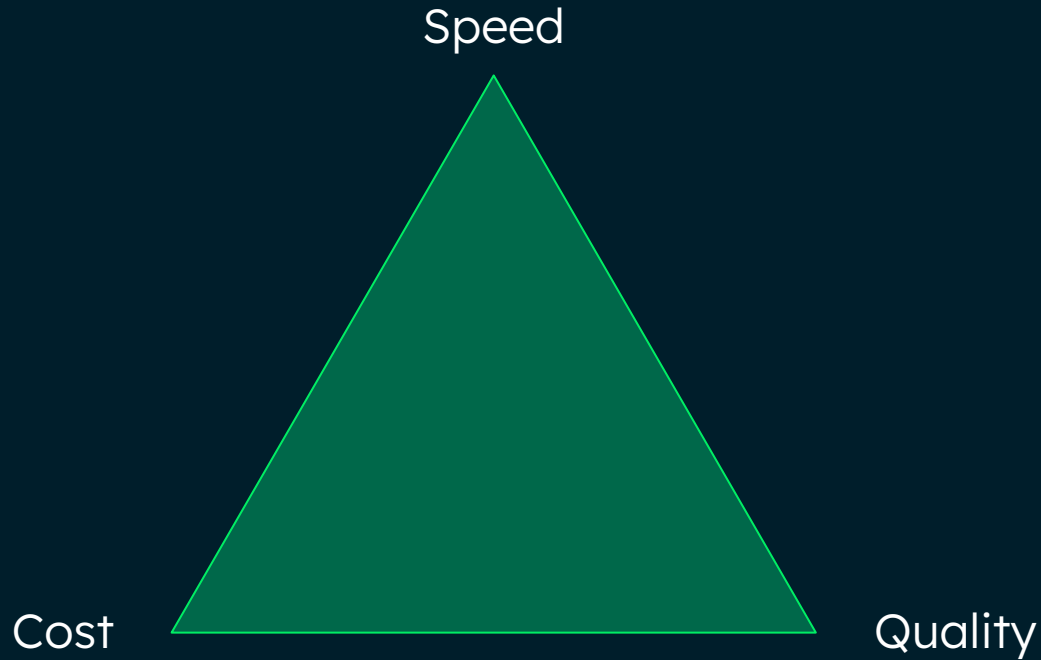
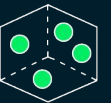
Max Scale before Sharding*

No Quantization + low-cpu	Binary Quantization + high-cpu	Binary Quantization + storage-optimized
130M	220M	410M

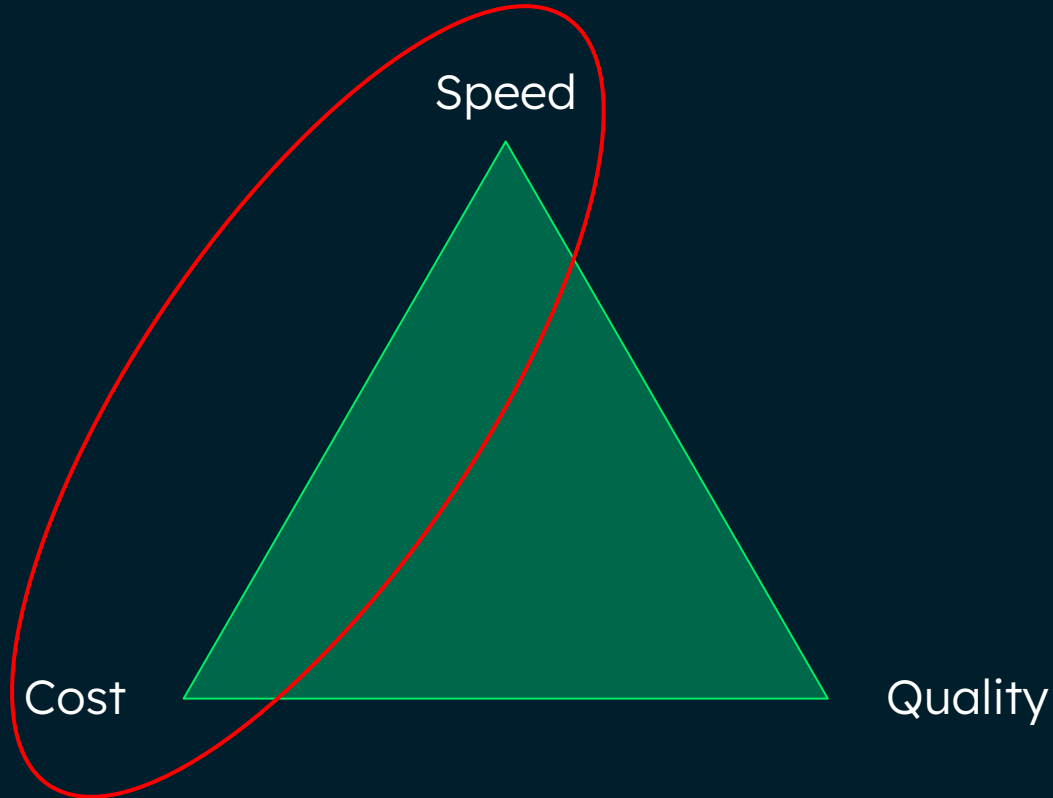
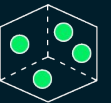
* Sized using 1536d vectors



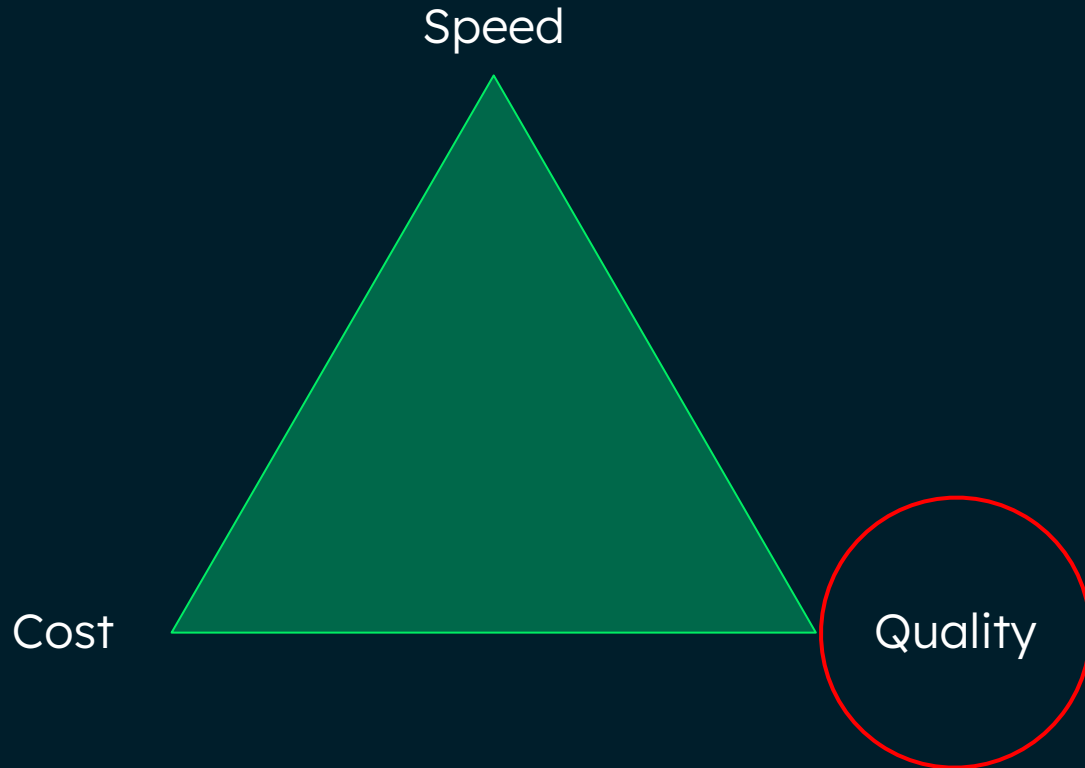
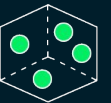
The Iron Triangle of Search



The Iron Triangle of Search

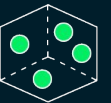


The Iron Triangle of Search





Vector search accuracy is often limited by the
embedding model you choose.





 MongoDB. + VOYAGE AI

Voyage AI's models



Embedding Models

General-Purpose

Text

Multimodal

Domain-Specific

Code

Legal

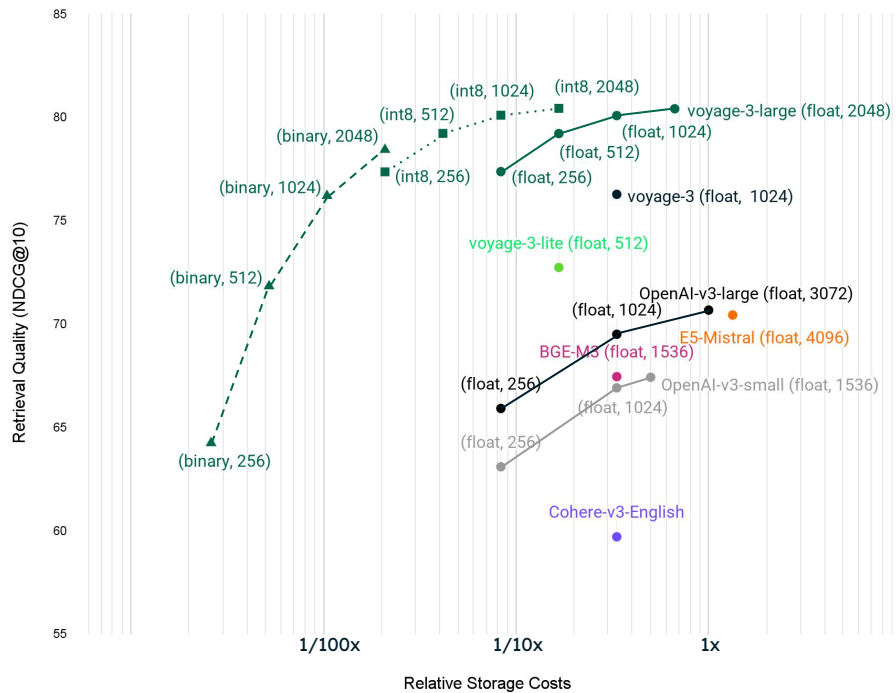
Finance

Rerankers

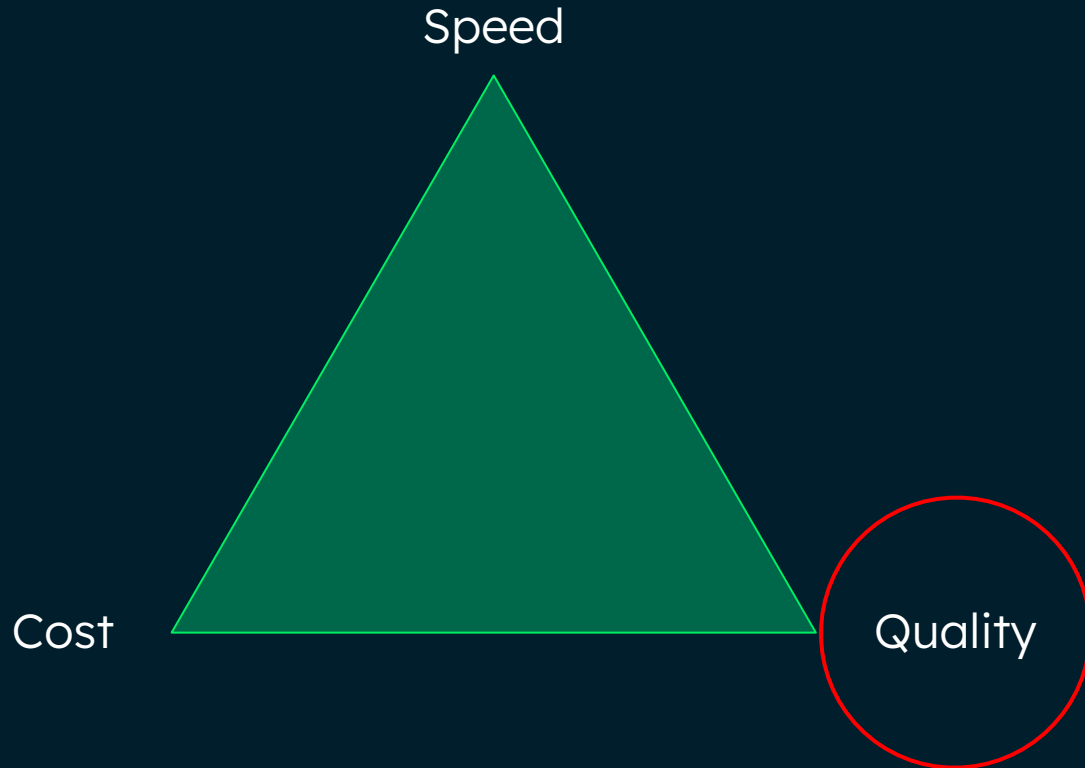
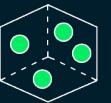
Standard

Lite

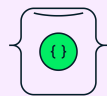
Voyage AI Embeddings Outperform



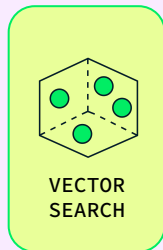
The Iron Triangle of Search



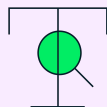
Integrating \$vectorSearch into MongoDB



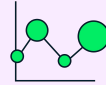
OLTP



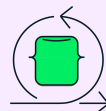
VECTOR
SEARCH



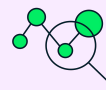
FULL-TEXT
SEARCH



ANALYTICS



STREAM
PROCESSING



TIME
SERIES



EMBEDDING
MODELS

Multi-Cloud Scale, Resilience, Performance, & Security

Holiday Apartment Marketplace

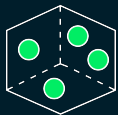


Sample Document



```
{
  "name": "Private Room In Bushwick",
  "description": "Here exists a very cozy room for rent in a
    shared 4-bedroom apartment.",
  "description_embedding": [0.43, -0.21, 0.11, 0.06, ...],
  "neighborhood_overview": "Bushwick is a northern Brooklyn
    neighborhood in New York City that has transformed...",
  "number_of_reviews": 1,
  "address": {
    "street": "Brooklyn, NY, United States",
    "suburb": "Brooklyn",
    "government_area": "Bushwick",
    "market": "New York",
    "country": "United States",
    "country_code": "US",
  },
  "location": {
    "type": "Point",
    "coordinates": [-73.93615, 40.697791],
    "is_location_exact": true,
  },
  "_id": 0,
  "review_score": 100,
}
```

Hybrid Search via \$rankFusion



Vector Search

Works well with open-ended or imprecise questions

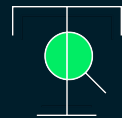
Highly efficient for high-dimensional data (docs, images, audios, videos)

Hybrid Search

Improved accuracy

Efficiently handles complex data

More robust search experience



Text Search

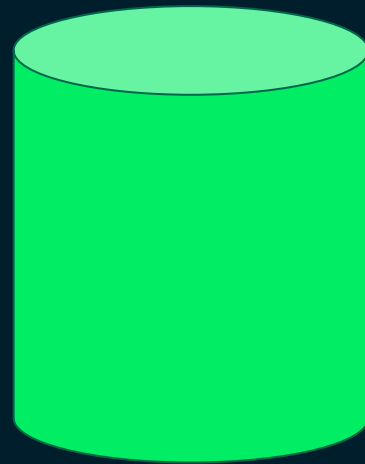
Fast and efficient for finding exact matches

Requires less computational resources and costs

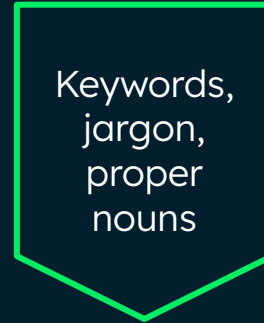
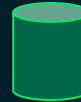
Full-text / lexical relevance via \$search



Out of sample terms



voyage-3-large
training dataset

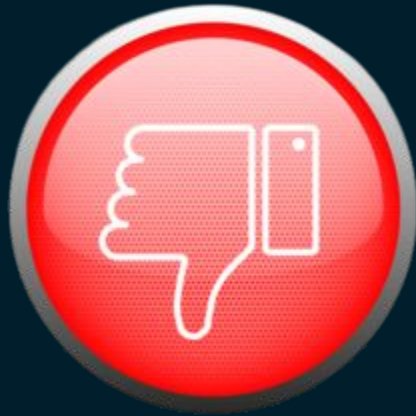


my dataset

Geospatial relevance via \$geoNear



Proxies of relevance via \$sort/\$match

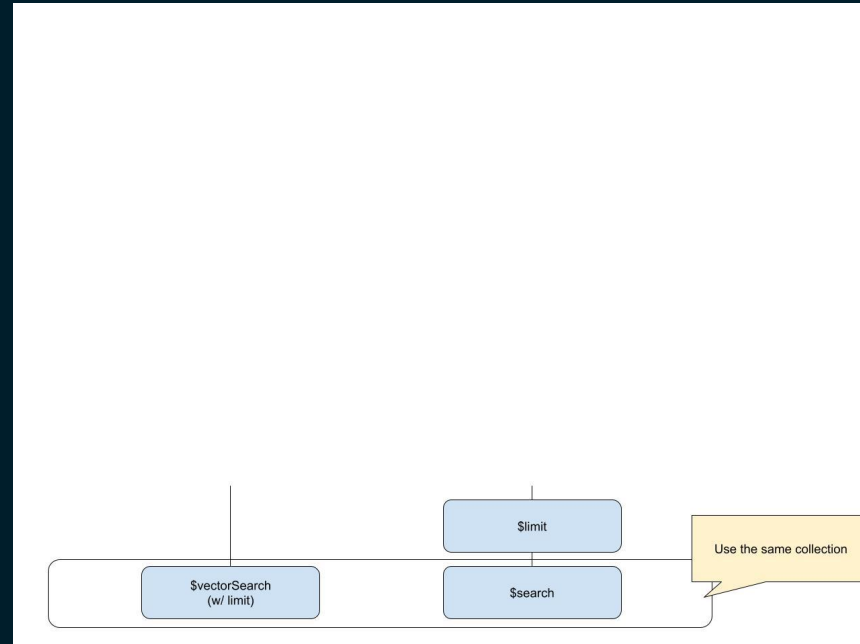


Current State of Hybridizing with \$vectorSearch

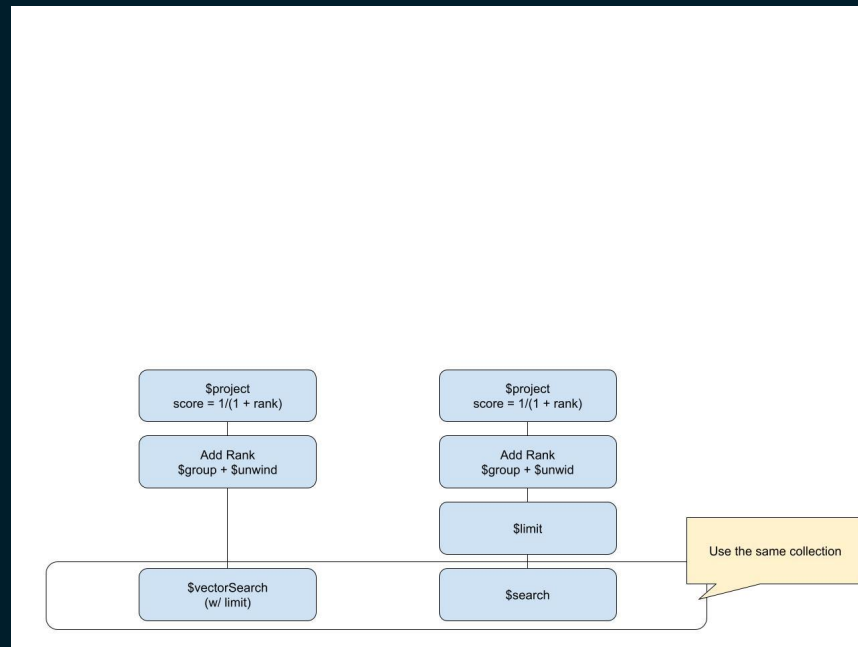


\$unionWith + \$group

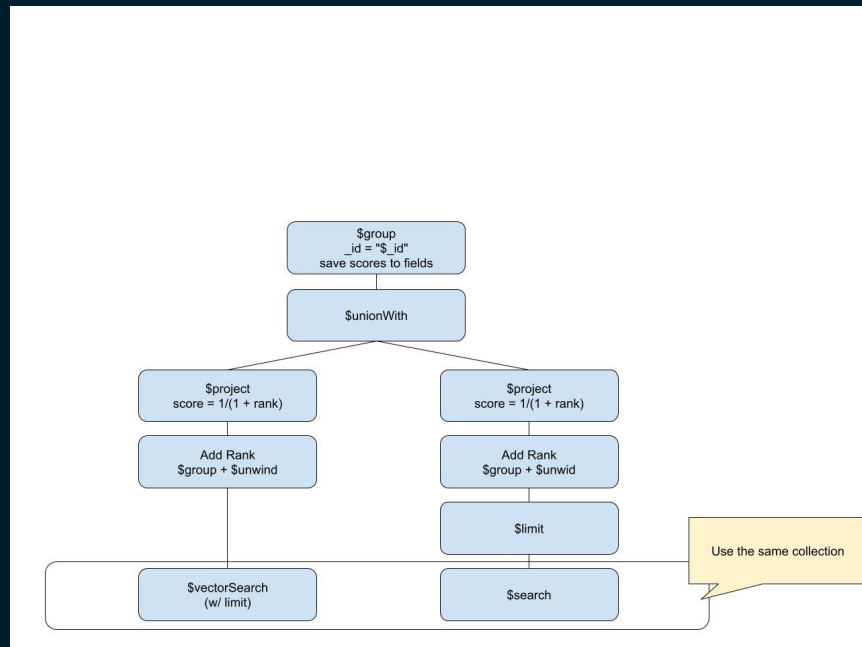
Current State of Hybridizing with \$vectorSearch



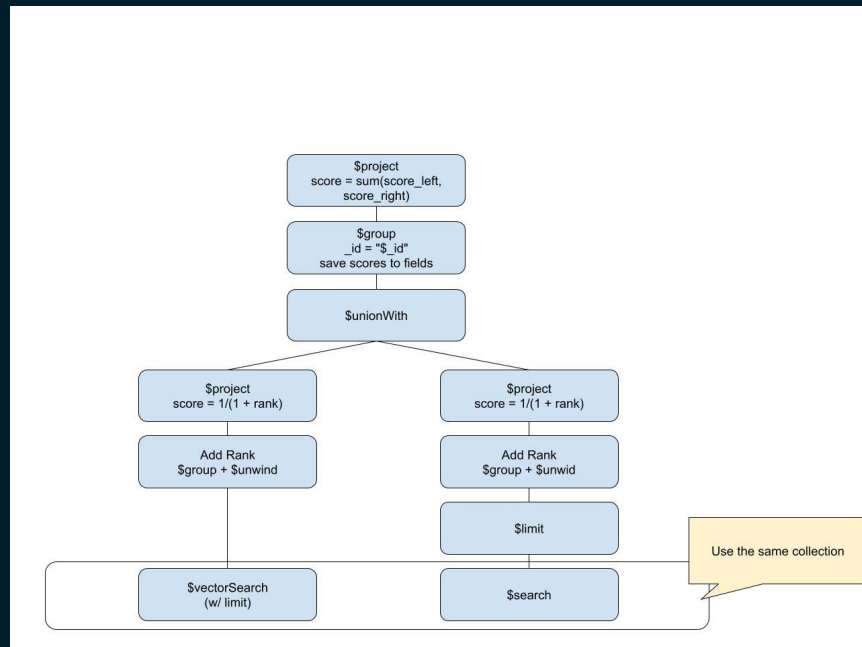
Current State of Hybridizing with \$vectorSearch



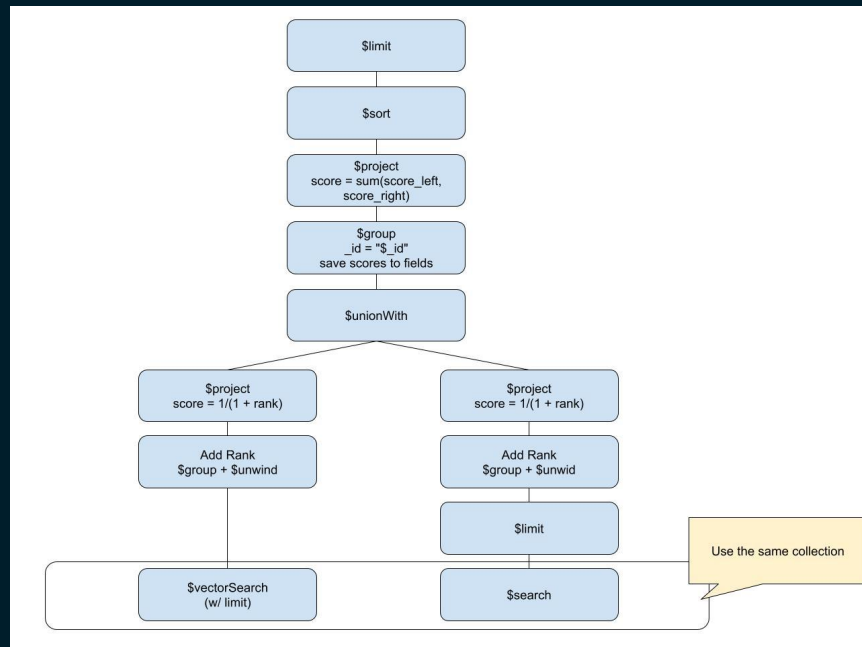
Current State of Hybridizing with \$vectorSearch



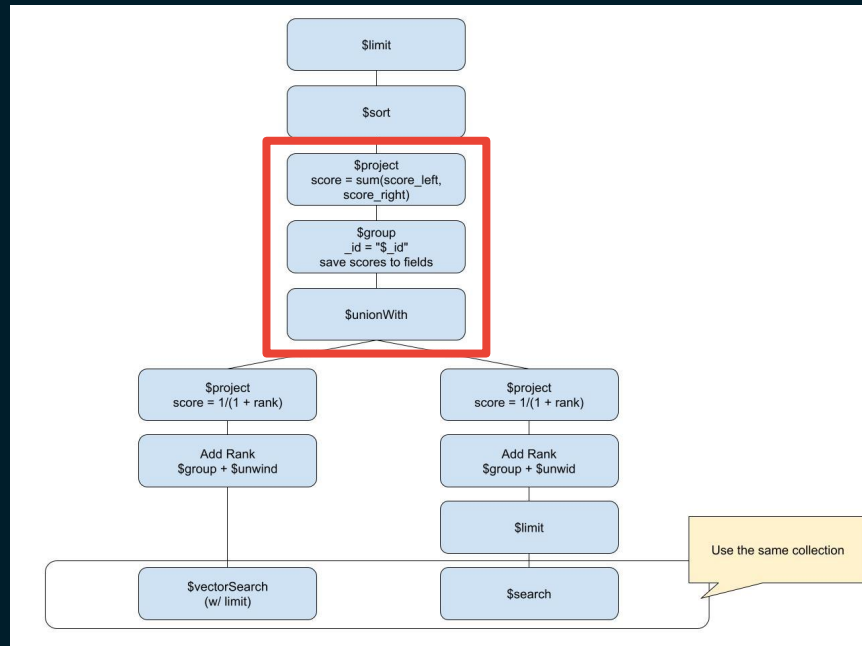
Current State of Hybridizing with \$vectorSearch



Current State of Hybridizing with \$vectorSearch



Current State of Hybridizing with \$vectorSearch



Reciprocal Rank Fusion Query Pipeline

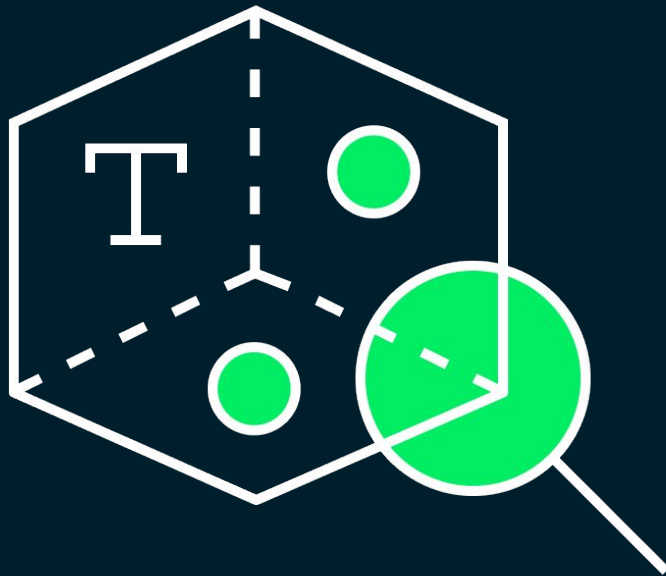


> Stage 1	<input type="text" value="\$vectorSearch"/>	☒	Perform vector search
> Stage 2	<input type="text" value="\$group"/>	☒	Group vector results
> Stage 3	<input type="text" value="\$unwind"/>	☒	Unwind to get ordinal rank
> Stage 4	<input type="text" value="\$addFields"/>	☒	Calculate reciprocal rank
> Stage 5	<input type="text" value="\$project"/>	☒	Select output fields
> Stage 6	<input type="text" value="\$unionWith"/>	☒	Combine with text search pipeline (includes group/unwind)
> Stage 7	<input type="text" value="\$group"/>	☒	Group by unique result document ID
> Stage 8	<input type="text" value="\$project"/>	☒	Select output fields
> Stage 9	<input type="text" value="\$addFields"/>	☒	Calculate score fusion
> Stage 10	<input type="text" value="\$sort"/>	☒	Sort by new score
> Stage 11	<input type="text" value="\$limit"/>	☒	Take top N

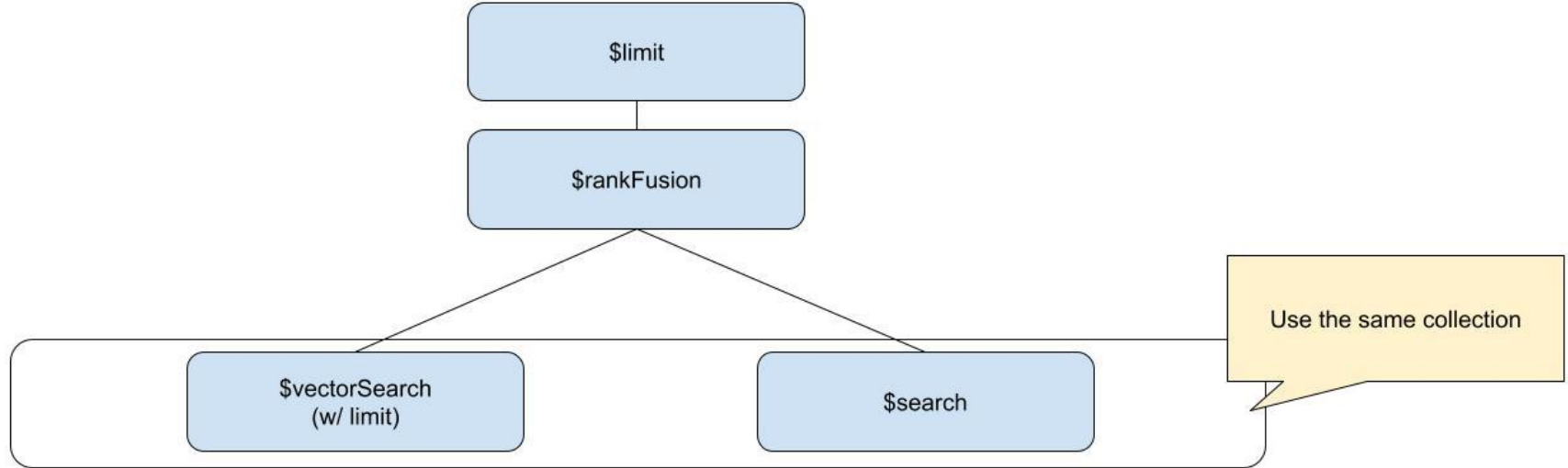
This is cool, but...



Introducing \$rankFusion on 8.0



Hybrid Search via \$rankFusion



Hybrid Search via \$rankFusion

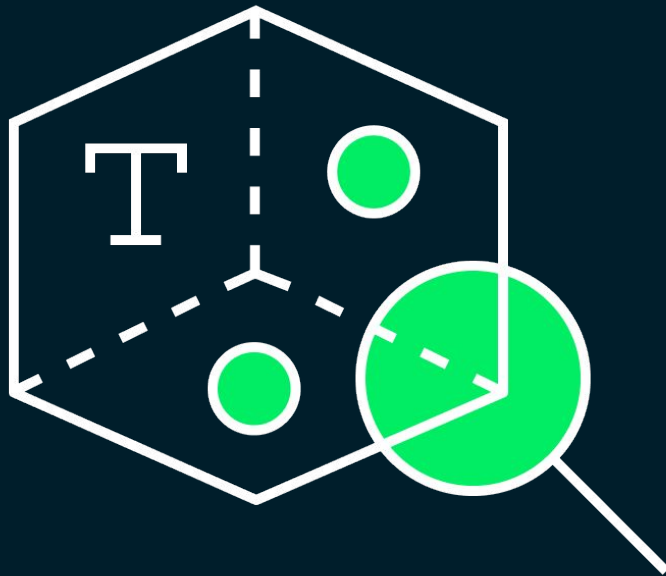


```
{ $rankFusion: {  
  input: {  
    pipelines: {  
      <myPipeline1>: <expression>,  
      <myPipeline2>: <expression>,  
      ...  
    }  
  },  
  combination: {  
    weights: {  
      <myPipeline1>: <numeric expression>,  
      <myPipeline2>: <numeric expression>,  
      ...  
    }  
  },  
  scoreDetails: <bool>  
} }
```

sky cinema



Introducing \$scoreFusion on 8.2

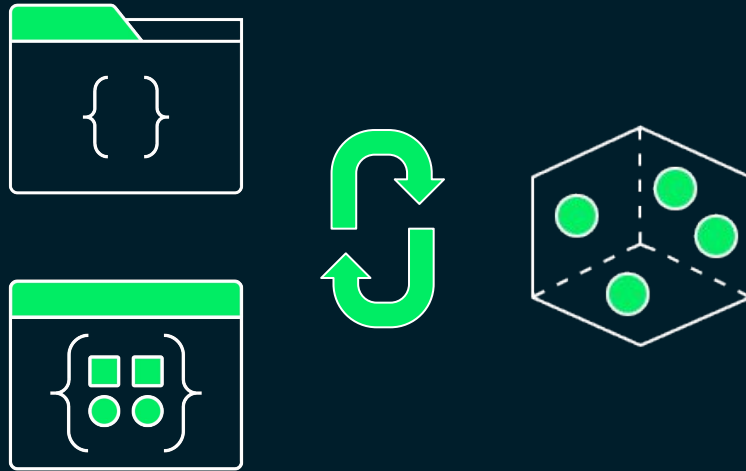


Hybrid Search via \$scoreFusion



```
{ $scoreFusion: {  
  input: {  
    pipelines: {  
      <input-pipeline-name>: <expression>,  
      <input-pipeline-name>: <expression>,  
      ...  
    },  
    normalization: "none|sigmoid|minMaxScaler"  
  },  
  combination: {  
    weights: {  
      <input-pipeline-name>: <numeric expression>,  
      <input-pipeline-name>: <numeric expression>,  
      ...  
    },  
    method: "avg|expression",  
    expression: <expression>  
  }  
} }
```

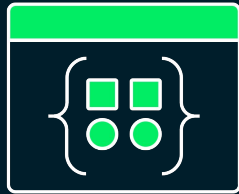
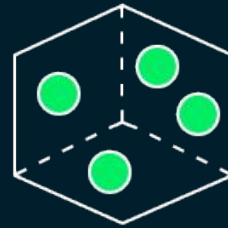
Introducing of Indexing on Views



Introducing of Indexing on Views



\$match
\$addFields
\$set

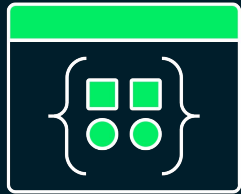
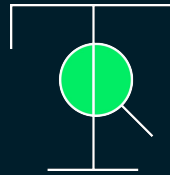


\$vectorSearch

Introducing of Indexing on Views



\$match
\$addFields
\$set



\$search

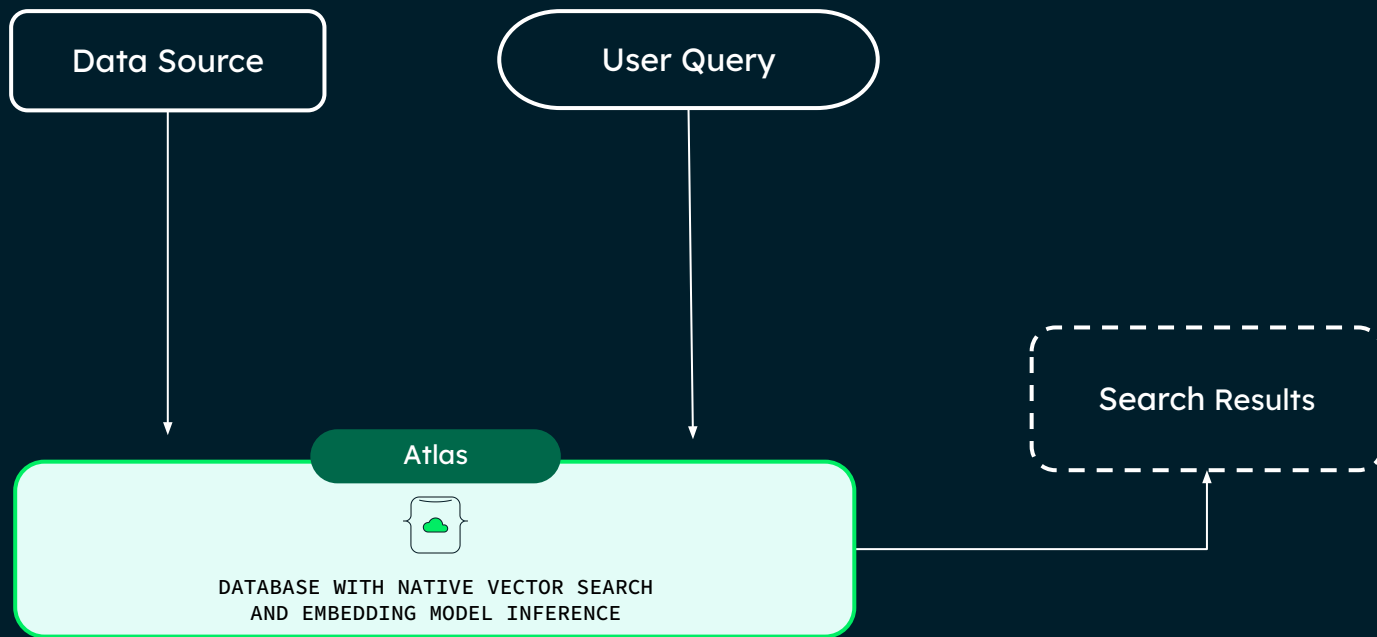
Indexing on Views (GA)



```
db.createView(  
  "512_embeddings",  
  "wikipedia-22-12-en 2",  
  [  
    {  
      $addFields: {  
        "512_embedding": { $slice:  
          ["$embedding", 512] }  
      }  
    }  
  ]  
);
```

```
db.512_embeddings.createSearchIndex(  
  "my_vector_index",  
  "vectorSearch",  
  {  
    "fields": [ {  
      "type": "vector",  
      "path": "embedding",  
      "numDimensions": 1024,  
      "similarity": "dotProduct"  
    },  
    {  
      "type": "vector",  
      "path": "512_embedding",  
      "numDimensions": 512,  
      "similarity": "cosine"  
    }  
  ]  
}  
);
```

Semantic search via Auto-embedding



Vector embeddings are generated and stored as part of Vector search index

Auto embedding index definition



'Bring your own vector' index definition	Auto embedding index definition
<pre>{ type: "vectorSearch", definition: { "fields": [{ "type": "vector", "numDimensions": 1536, "similarity": "cosine", "path": "content_embeddings" }] } }</pre>	<pre>{ type: "vectorSearch", definition: { "fields": [{ "type": "text", "model": "voyage-3-large", "path": "content", }] } } // ** the naming/syntax of the fields can change in // Public preview and beyond</pre>

Auto embedding query



'Bring your own vector' query syntax	Auto embedding query syntax
<pre>{ "\$vectorSearch": { "index": "indexName", "queryVector": [0.03898080065846443, -0.05879044905304909, ...], "path": "content_embeddings", "limit": 5 "numCandidates": 50 } }</pre>	<pre>{ "\$vectorSearch": { "index": "indexName", "query": "<user-query>", "path": "content", "limit": 5 "numCandidates": 50, }, }</pre>

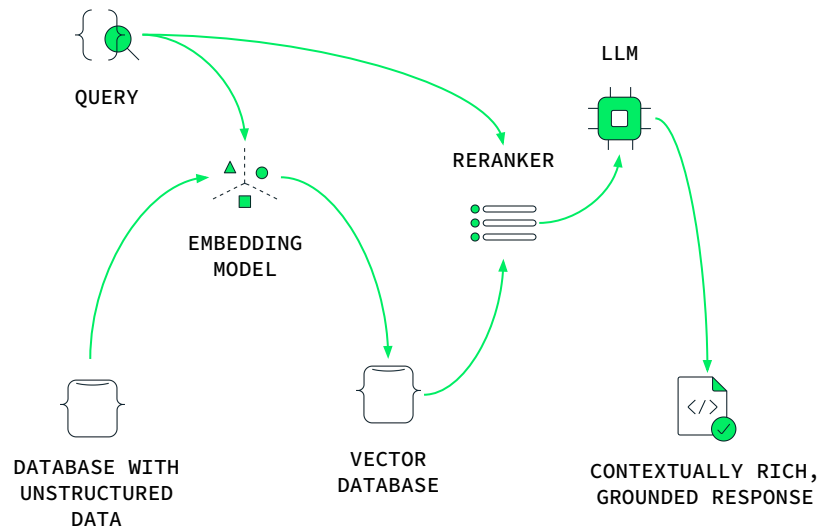
Greater Seamlessness in MongoDB 8.x



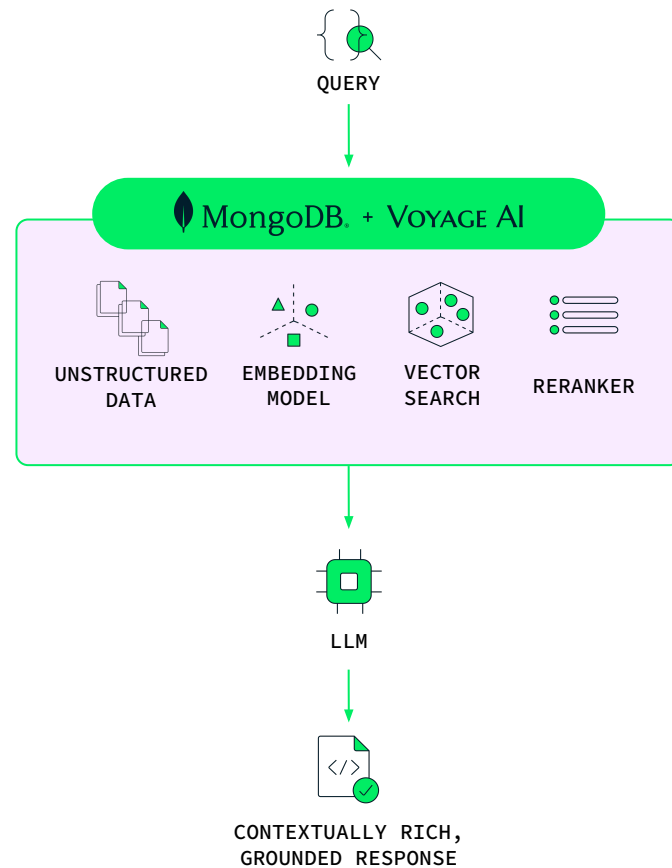
Greater Seamlessness in the Future



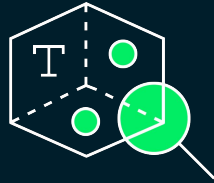
BEFORE



AFTER



Greater Seamlessness in the Future





①

To meet the needs of **next generation workloads**, we must provide a **vector search capability** that is seamless to the **rest of MongoDB**.

②



③





Thank you