

Retrieval Augmented Generation



Context Windows

- AI models have context windows that limit how much text you can feed them
 - Ex) Gemini 3 Flash and Pro have 1 million token context window – about 1,500 pages of text
- Even if you can fit your documents in that window, the AI can get “lost in the middle” and have trouble answering your queries
- We need a way for the AI to intelligently select the relevant parts of your data in order to answer your question

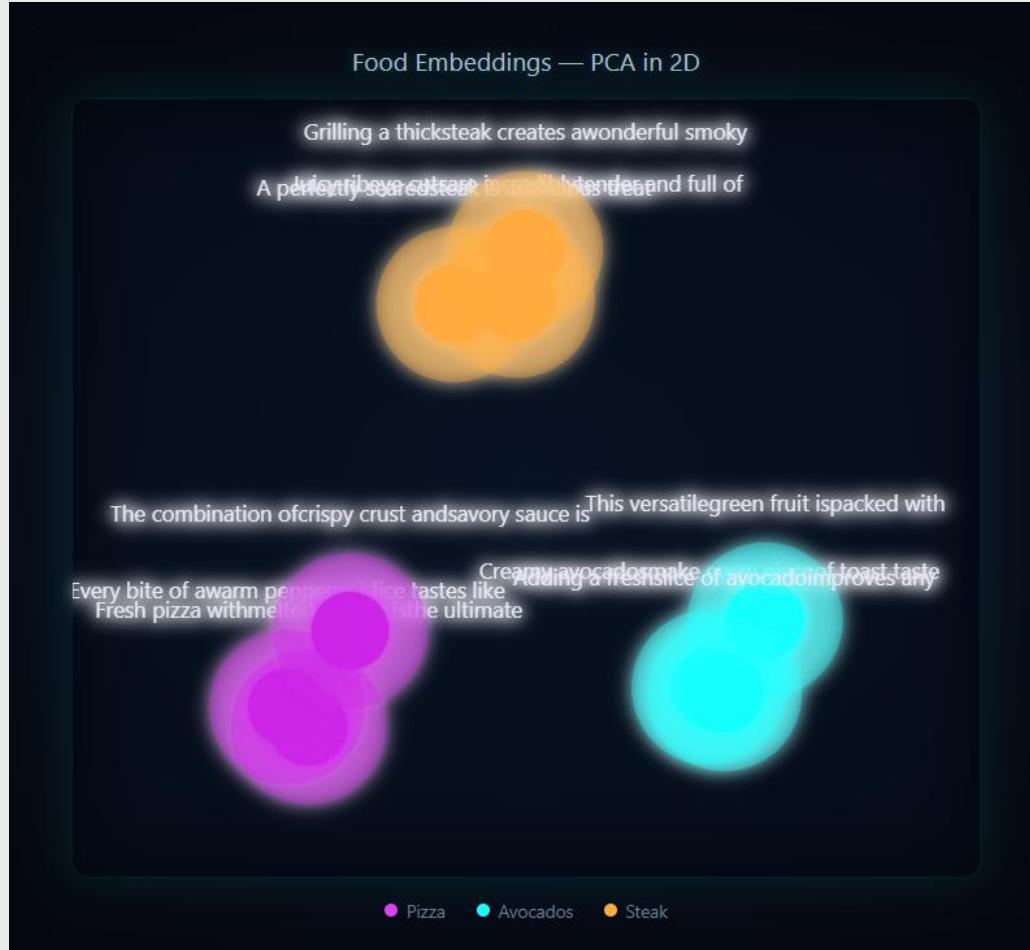
Retrieval Augmented Generation (RAG)

- **RAG allows an AI to selectively retrieve documents to answer your query**
- **All documents stored in a clever way in a special database**
- **This database uses clever techniques to find relevant documents for your query**

Text Embeddings

- The key to RAG are AI powered text embeddings
- A text embedding maps text to a vector
- The vector location encodes the meaning of the text

Embedding Example



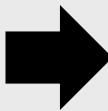
Embedding Models

- We can use the gemini-embedding-001 model to embed text
- Other AIs have similar embedding models
- Embedding is 768 dimensions
- Python code:

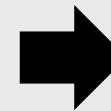
```
def get_embedding(text: str, client) -> list:  
    """Generate 768-dim embedding (matches chat.py / rag_ingest.py style)."""  
    result = client.models.embed_content(  
        model="gemini-embedding-001",  
        contents=text,  
        config=types.EmbedContentConfig(  
            task_type="RETRIEVAL_DOCUMENT",  
            output_dimensionality=768,  
        ),  
    )  
    return list[Any](result.embeddings[0].values)
```

Embedding Chunks

- We chop up the document in smaller chunks (maybe 1 or 2 pages)
 - Make chunks overlap a little bit so you don't cut important parts in the middle
- We embed each chunk one at a time
- Now we have a bunch of vectors, where do we store them?



```
def get_embedding(text: str, client) -> list:  
    """Generate 768-dim embedding (matches chat.py / rag_ingest.py style)."""  
    result = client.models.embed_content(  
        model="gemini-embedding-001",  
        contents=text,  
        config=types.EmbedContentConfig(  
            task_type="RETRIEVAL_DOCUMENT",  
            output_dimensionality=768,  
        ),  
    )  
    return list[Any](result.embeddings[0].values)
```



Vector Store

- A vector store is a database where can store the text chunks and their vector embedding
- Vector store also has methods to let you rapidly search for vectors which are similar to an input vector
 - Hierarchical navigable small world (HNSW)
 - Inverted File Index (IVF)
- This lets us quickly find chunks similar to our query
- Many popular vector stores
 - Pinecone
 - Chroma/Weaviate
 - Milvus
 - MongoDB ☺

RAG Workflow

- **User query**
- **Embed query**
- **Search vector store for chunks similar to query**
- **Put the chunk text into the chat context**
- **AI replies to your query + relevant chunks**

Whats so special about pizza?



$$[x_1 \ x_2 \ \dots \ x_m]$$



Whats so special about pizza? + Pizza is delicious and satisfying + Nothing beats a hot slice of pizza + Pizza brings people together like nothing else



Pizza is tasty and brings us together



Embedding Cost

- Embeddings are very cheap

Google Gemini Pricing (February 2026)

| Tier | Price per 1 Million Tokens | Rate Limits |
|-----------|----------------------------|---|
| Free Tier | \$0.00 | Up to 1,500 requests per day (RPD). |
| Paid Tier | \$0.15 | Up to 5,000,000 tokens per minute (TPM). |
| Batch API | \$0.075 (50% discount) | Optimized for massive, asynchronous datasets. |

OpenAI Embedding Pricing (February 2026)

| Tier | Price per 1 Million Tokens | Rate Limits (Tier 3 Example) |
|--------------------|---------------------------------|---|
| Standard (3-small) | \$0.02 | Up to 5,000,000 tokens per minute (TPM). |
| Standard (3-large) | \$0.13 | Up to 5,000,000 tokens per minute (TPM). |
| Batch API | \$0.01 – \$0.065 (50% discount) | Optimized for massive, non-urgent datasets. |

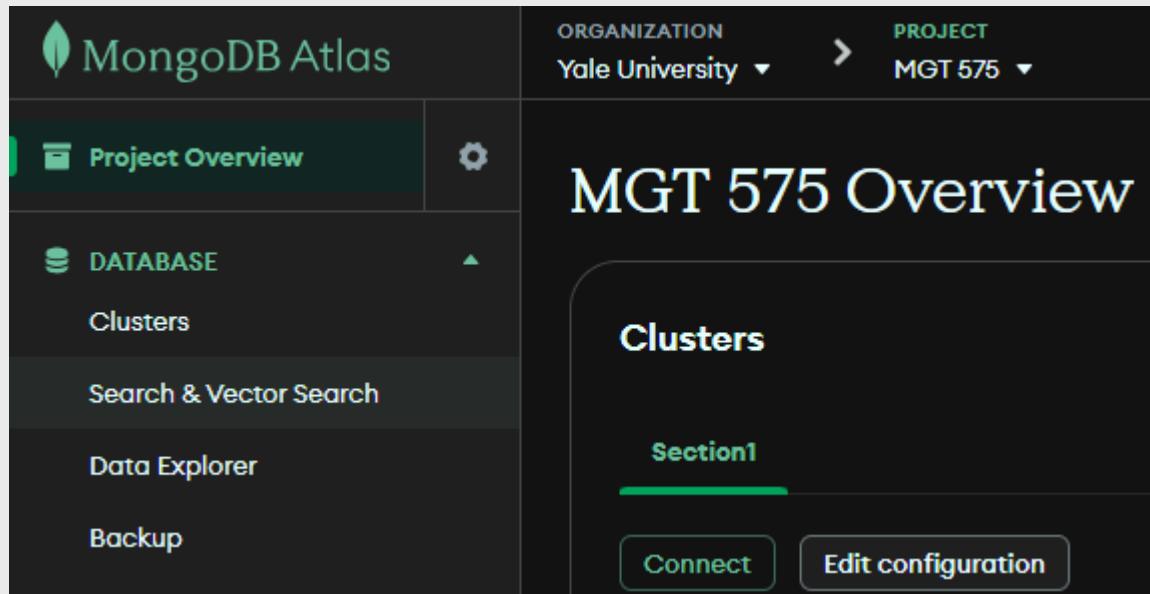
MongoDB Vector Store

- After we embed the chunks with AI, we store them in a collection on MongoDB
 - Chunk text
 - Embedding vector
 - Useful metadata (can be AI generated, like summarize the chunk)

```
_id: ObjectId('699bc2d4145c585e9554853b')
caseName: "Nathan's Famous"
pageNumber: 1
content: "<p>Yale SCHOOL OF MANAGEMENT</p>
<p>YALE CASE 20-020 JULY 4, 2020</p>
..."
summary: "The case examines a dramatic shift on Nathan's Famous's balance sheet,..."  
▼ content_vector : Array (768)
  0: 0.006767863
  1: 0.003748851
  2: 0.01138478
  3: -0.061705638
  4: -0.0113745965
  5: 0.017322833
  6: -0.012120146
  7: -0.016870946
  8: -0.0032352665
  9: 0.0010216809
  10: 0.0051032566
  11: -0.011265037
  12: -0.0036041876
  13: 0.004373133
  14: 0.09579993
  15: 0.007351367
  16: 0.0039968383
  17: -0.0002347112
  18: -0.0022181787
  19: -0.01734156
  20: 0.0021275913
  21: 0.015317333
  22: 0.0048162555
  23: 0.0052397493
  24: 0.031737395
▼ Show 743 more fields in content_vector
```

MongoDB Vector Store

- Once the chunks are stored, we create an **index** on the collection
- Select “Search & Vector Search”



MongoDB Vector Store

- Choose “Vector Search” for search type

The screenshot shows a dark-themed wizard titled "Create a Vector Search Index" with three steps: "Setup" (highlighted in green), "Configuration", and "Review". The "Setup" step is titled "Start Your Index Configuration" and contains a "Search Type" section. It asks "Which Search type should I use?" and lists two options: "Atlas Search" (described as full-text search for relevance-based app features) and "Vector Search" (described as for semantic search and AI applications). The "Vector Search" option is highlighted with a green border.

1
Setup

2
Configuration

3
Review

Create a Vector Search Index

Start Your Index Configuration

Search Type
Which Search type should I use?

Atlas Search
Full-text search for relevance-based app features.

Vector Search
For semantic search and AI applications.

MongoDB Vector Store

- Name your index and choose the database and collection where you stored your chunks

Index Name and Data Source
Search indexes are specific to a database and collection.

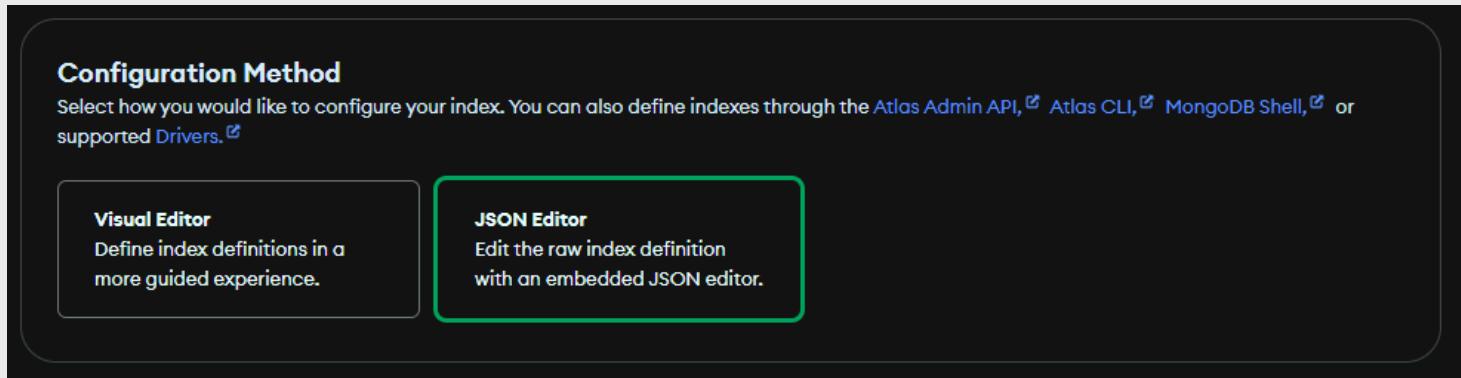
Index Name
vector_index

Database and Collection
Search for database or collection ...

- case_blind_elephant
- case_nathans
- chatapp

MongoDB Vector Store

- Choose “JSON Editor” for configuration method



- Have the AI write the JSON index definition and paste it in the JSON Editor on MongoDB

```
{  
  "fields": [  
    {  
      "type": "vector",  
      "path": "content_vector",  
      "numDimensions": 768,  
      "similarity": "cosine"  
    }  
  ]  
}
```

Coding Session

- Put a large document into a vector store
- Clone a basic chat app
- Give that chat bot RAG functionality