Vector-Based Semantic Search and RAG with Apache Beam

Claude van der Merwe

Agenda

1. **Introduction**
2. **Semantic Search Fundamentals**
3. **Vector databases**
4. **Retrieval Augmented Generation (RAG)**
5. **Apache Beam RAG module**
6. **Demo**

Why Vector Search & RAG Matter

Traditional Search Limitations:

- Keyword matching misses semantic meaning
- Synonym problems ("car" vs. "automobile")
- Language barriers and translation issues
- Inability to understand context

LLM Challenges:

- Knowledge cutoff limitations
- Potential for hallucinations
- No access to private/proprietary data
- Costly to fine-tune on new information

The Solution:

- Vector search finds semantically related content
- RAG combines retrieval with generation
- Grounded responses with up-to-date information

Semantic Search Fundamentals: Embeddings

What are Embeddings?

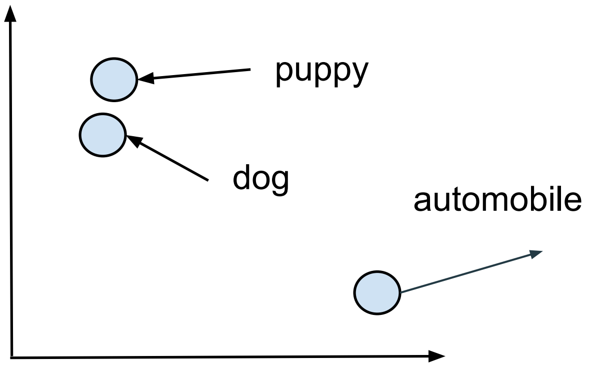
- Mathematical representation of meaning
- Dense vectors with hundreds of dimensions
- Convert text and images into numerical form
- Similar meanings have similar vector representations

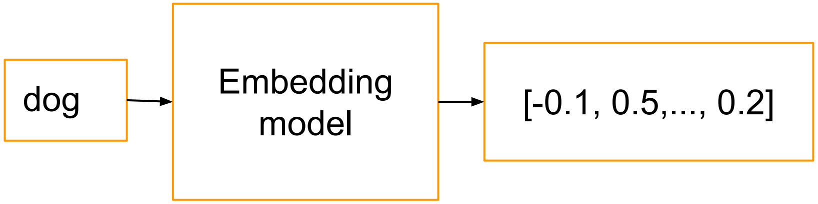
How Embeddings Work:

- Words/concepts with related meanings cluster together
- Mathematical distance = semantic distance
- Trained on vast amounts of text to capture relationships

Embedding Models:

- Commercial: Vertex AI Gecko
- Open source: Hugging Face Sentence Transformers
- Custom: Fine-tuned for specific domains

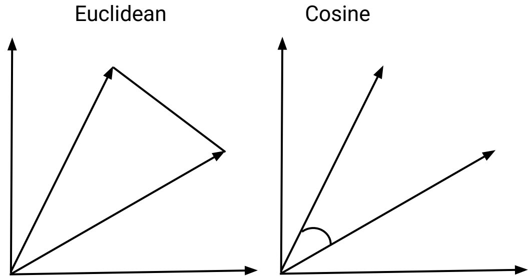




Semantic Search Fundamentals: Vector Similarity

Vector Similarity Metrics:

- Measuring how close vectors are in the embedding space
- Different metrics for different applications



Distance Metric	**Properties measured**	**Example use case**
Euclidean distance	Magnitude and direction	K-means clustering
Cosine similarity	Only Direction	Text similarity
Dot product similarity	Magnitude and direction	Text similarity with normalized vectors

Note: All three methods provide same semantic search results for normalized embeddings.

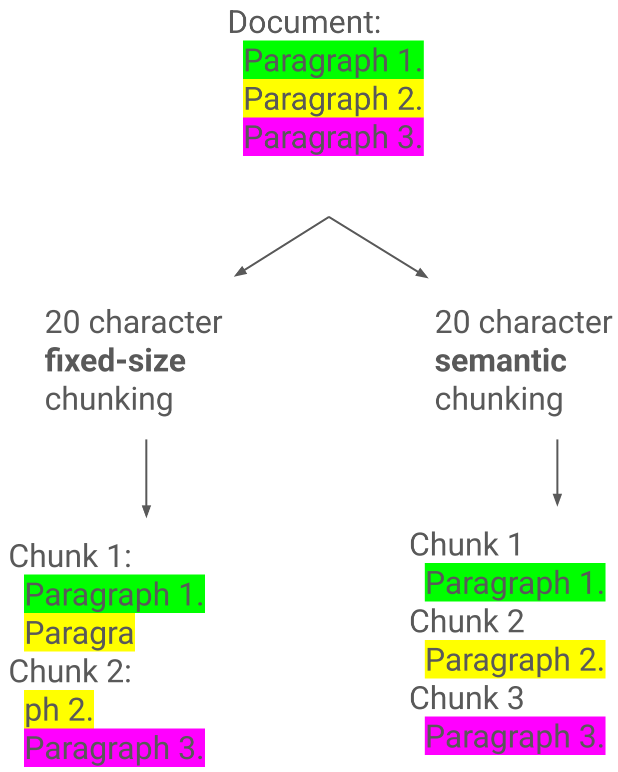
Semantic Search Fundamentals: Chunking

Why Chunking Matters:

- Documents often exceed embedding model context limits
- Smaller chunks enable more precise retrieval
- Chunk size affects relevance and performance

Basic Chunking Strategies:

- **Fixed-size chunking -** Split every N tokens/characters)
- **Semantic chunking -**Split at natural boundaries (paragraphs, sections)



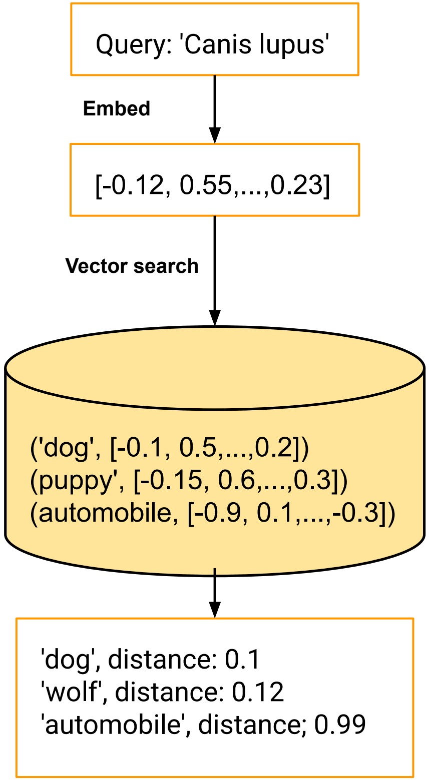
Vector Databases

The Role of Vector Databases:

- Specialized storage for embedding vectors
- Bridge between raw data and retrieval system
- Enable efficient semantic search at scale
- Support for multiple distance metrics (cosine, Euclidean)
- Metadata storage alongside vectors
- Filtering and hybrid search capabilities

Vector Database Operations:

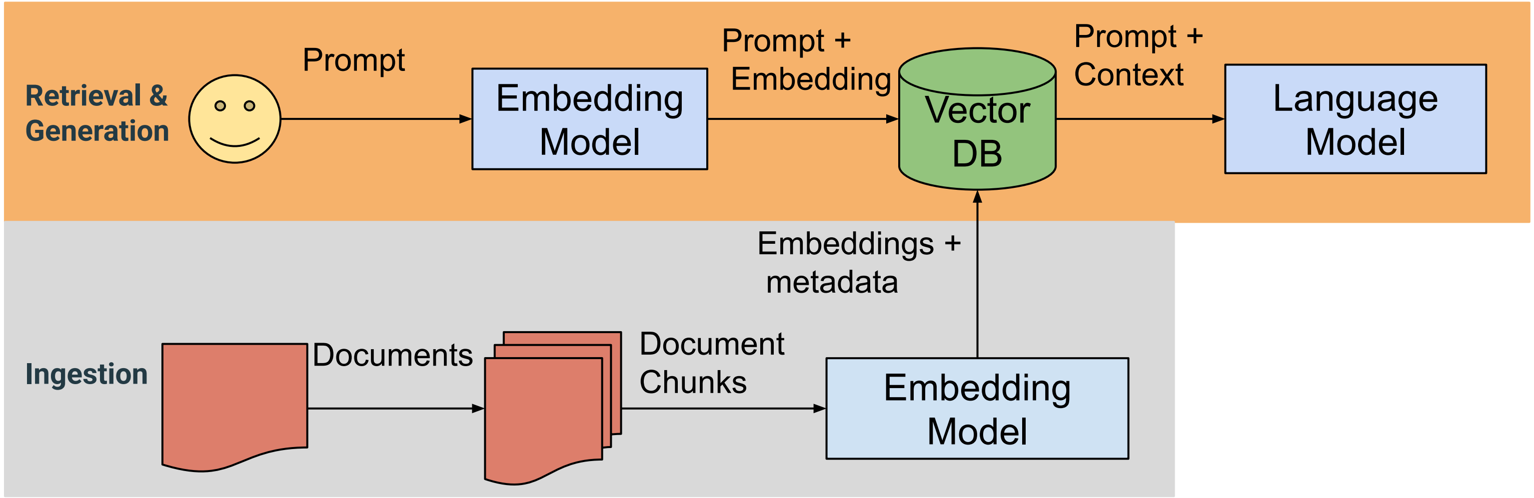
- **Indexing**: Organize vectors for efficient retrieval
- **Search**: Find nearest neighbors to query vector
- **Filtering**: Combine vector search with metadata filtering
- **Updates**: Maintain freshness of vector collections



Retrieval Augmented Generation (RAG): Overview

What is RAG?

- Uses semantic search to enrich LLMs with external knowledge
- Creates grounded, accurate AI responses
- Bridges the gap between static LLM knowledge and fresh data



Apache Beam RAG Module: Extensible Embedding Transforms

- Chunking - `apache_beam.ml.rag.chunking`
  - Supports LangChain package chunking
  - Implement your own chunking strategies
- Embedding - `apache_beam.ml.rag.embeddings`
  - Supports local Hugging Face sentence-transformers
  - Supports remote Vertex AI embedding models
  - Easily add new embedding models
- Ingestion - `apache_beam.ml.rag.ingestion`
  - Supports BigQuery and Postgres
  - Integrate with your favorite vector database
- Enrichment - `apache_beam.ml.rag.enrichment`
  - Supports BigQuery only
  - Implements Enrichment Transforms for performing Similarity search

Apache Beam RAG module: Chunk class

Chunk represents a unit of embeddable content used throughout embedding generation, ingestion and vector search.

Data to be embedded

```
@dataclass
class Chunk:
  content: Content
  id: str
  index: int = 0
  metadata: Dict[str, Any]
  embedding: Optional[Embedding]
```

Ingested data

```
{
  "id": "desk-001",
  "name": "Modern Desk",
  "description": "Sleek...",
  "category": "Desks"
}
```

↓

```
Chunk(
 id="desk-001",
 content=Content(
  text="Modern Desk Sleek…"
 ),
 index=0
 metadata={'category': "Desks",
…}
)
```

Apache Beam RAG module: LangChainChunker

Input:

```
{
  'content': 'This is a simple test document. It has multiple sentences.',
  'source': 'simple.txt',
  'language': 'en'
}
```

Output:

```
Chunk(
 content='This is a simple test document',
 index=0,
 metadata={'source': 'simple.txt', 'language': 'en'},
 id='simple.txt_0'
)
Chunk(
 content='It has multiple sentences',
 index=1,
 metadata={'source': 'simple.txt', 'language': 'en'}
 id='simple.txt_1'
)
```

Code snippet:

```
from apache_beam.ml.transforms.base import MLTransform
from apache_beam.ml.rag.chunking.langchain import LangChainChunker
from langchain.text_splitter import RecursiveCharacterTextSplitter


# ... pipeline code
"Chunk document" >> MLTransform().with_transform(
  LangChainChunker(
    text_splitter=RecursiveCharacterTextSplitter(
      chunk_size=50,
      chunk_overlap=0,
      separators=["."]
    ),
    document_field="content",
    metadata_fields=["source", "language"],
    chunk_id_fn=lambda x: f"{x.metadata['source']}_{x.index}"
  )
)
# ... pipeline code
```

Apache Beam RAG module: Embeddings

Input:

```
Chunk(
 content='This is a simple test document',
 index=0,
 metadata={'source': 'simple.txt', 'language': 'en'},
 id='simple.txt_0'
)
Chunk(
 content='It has multiple sentences',
 index=1,
 metadata={'source': 'simple.txt', 'language': 'en'},
 id='simple.txt_1'
)
```

Code snippet:

```
from apache_beam.ml.transforms.base import MLTransform
from apache_beam.ml.rag.embeddings.huggingface import HuggingfaceTextEmbeddings


# ... pipeline code
'Generate Embeddings' >> MLTransform()
  .with_transform(
  HuggingfaceTextEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2")
)
# ... pipeline code
```

Output:

```
Chunk(
 content='This is a simple test document',
 …
 id='simple.txt_0',
 embedding=[0.5, 0.6, 0.7]
)
Chunk(
 content='It has multiple sentences',
 …
 id='simple.txt_1',
 embedding=[0.1, 0.2, 0.3]
)
```

Apache Beam RAG module: Ingestion

Input:

```
Chunk(
 content='This is a simple test document' ,
 index=0,
 metadata={'source': 'simple.txt', 'language': 'en'},
 id='simple.txt_0' ,
 embedding=[0.5, 0.6, 0.7]
)
Chunk(
 content='It has multiple sentences' ,
 index=1,
 metadata={'source': 'simple.txt', 'language': 'en'}
 id='simple.txt_1'
 embedding=[0.1, 0.2, 0.3]
)
```

BigQuery table: document_embeddings

content	embedding	id	metadata
This is a simple test document	[0.5,0.2...]	simple.txt_0	{language:en...}
It has multiple sentences	[0.2,0.3...]	simple.txt_1	{language:en...}

Code snippet:

```python
from apache_beam.ml.rag.ingestion.bigquery import BigQueryVectorWriterConfig
from apache_beam.ml.rag.ingestion.bigquery import SchemaConfig

BigQueryVectorWriterConfig(
  write_config={
      'table': 'document_embeddings',
      'create_disposition': 'CREATE_IF_NEEDED',
      'write_disposition': 'WRITE_TRUNCATE'
  },
  # Optional
  schema_config=SchemaConfig(
      schema=<BigQuery schema dictionary>,
      chunk_to_dict_fn=chunk_to_dict
  )
)

# ... pipeline code
'Write to BigQuery' >> VectorDatabaseWriteTransform(bigquery_writer_config)
# ... pipeline code
```

Apache Beam RAG module: Semantic Search

Scenario - Consider an online store with:

1. A vector database containing product catalog embeddings
2. A streaming pipeline that processes product queries and returns relevant products

BigQuery table: product_catalog

id	embedding	description	price
laptop-001	[0.1,0.2...]	Powerful ultralight laptop	1999
desk-001	[0.3,0.4...]	Sleek modern desk	149
desk-002	[0.4,0.5...]	Vintage desk	300

Apache Beam RAG module: Semantic Search

Input:

```
{
 'query': 'powerful laptop for video editing',
'max_price': 2000
}
```

Output:

```
Chunk(
 content: 'powerful laptop for video editing',
 metadata: {
   'max_price': 2000,
   'enrichment_data': {
     'id': 'laptop-001',
     'description': 'Powerful ultralight laptop …',
     'price': 1999
   }
 }
 embedding = [...]
)
```

Code snippet:

```
from apache_beam.transforms.enrichment import Enrichment
from apache_beam.ml.rag.enrichment.bigquery_vector_search import (
    BigQueryVectorSearchParameters,
    BigQueryVectorSearchEnrichmentHandler
)

vector_search_params = BigQueryVectorSearchParameters(
    project='<project_id>',
    table_name='pduct_catalog',
    embedding_column="embedding",
    columns=["pice", "description"],
    metadata_restriction_template="price <= {max_price}"
    neighbor_count=1
)

pipeline
  | 'Read from PubSub' >> beam.io.ReadFromPubSub()
  | "Convert to Chunk" >> beam.Map(to_chunk)
  | 'Generate Embeddings' >> MLTransform()
    .with_transform(
      HuggingfaceTextEmbeddings(
        model_name="sentence-transformers/all-MiniLM-L6-v2"
      )
    )
  | 'Vector Search' >> Enrichment(
        BigQueryVectorSearchEnrichmentHandler(
        vector_search_parameters=vector_search_params,
        min_batch_size=1,
        max_batch_size=5
    )
  )
```

Demo

Links

- Code location and python doc
- Colabs
  - Vector Embedding Ingestion with Apache Beam and AlloyDB
  - Embedding Ingestion and Vector Search with Apache Beam and BigQuery

Thank you!