Lecture 20
Vector Databases

Prashant Pandey

prashant.pandey@utah.edu

Vector databases

- Specialized databases designed to store, index, and retrieve high-dimensional vectors efficiently
- Particularly useful for tasks like similarity search, recommendation systems, and AI model outputs

Metric-space vector databases

- These databases use distance metrics (e.g., Euclidean, cosine similarity) to organize and search vectors
- **Examples**:
    - Milvus
    - Weaviate
    - Pinecone

Graph-based vector databases

- Utilize graph structures (e.g., k-NN graphs, HNSW) for efficient similarity search
- These are well-suited for large-scale datasets where approximate nearest neighbor (ANN) searches are common
- **Examples**:
  - ElasticSearch (with ANN plugins)
  - Vespa
  - HNSWlib-based databases

Hash-based vector databases

- Use hashing techniques like Locality-Sensitive Hashing (LSH) for fast approximate searches
- Suitable for sparse or low-dimensional datasets.
- **Examples**:
  - FAISS (Flat and Hash-based indexing options)
  - Annoy (Approximate Nearest Neighbors)

Hybrid vector databases

- Combine vector indexing with traditional relational or document-based databases
- Ideal for applications needing structured data along with unstructured vector queries
- **Examples**:
  - Redis with vector similarity search
  - PostgreSQL with vector search extensions (e.g., pgvector)
  - MongoDB Atlas Search (supports vector fields)

Cloud-native vector databases

- Fully managed, scalable vector databases optimized for cloud platforms
- Simplify setup, scaling, and maintenance
- **Examples**:
  - Amazon Kendra
  - Google Vertex AI Matching Engine
  - Azure Cognitive Search

Specialized vector databases

- Tailored for specific use cases, such as video search, genomics, or geospatial data
- May incorporate domain-specific optimizations
- **Examples**:
  - Zilliz (AI and ML-focused)
  - Deep Lake (designed for AI datasets)

Vector embeddings

Vectors

- Commonly represent unstructured data
  - Audio, text, images, etc
- Usually of high-dimension in the form of a **dense** embedding.
- Packed with information
- Easy to use API to create

Vector Embedding Creation

- Simple creation APIs
- Example with HuggingFace Sentence Transformer

```python
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')

sentences = [
  "That is a very happy Person",
  "That is a Happy Dog",
  "Today is a sunny day"
]
embeddings = model.encode(sentences)
```

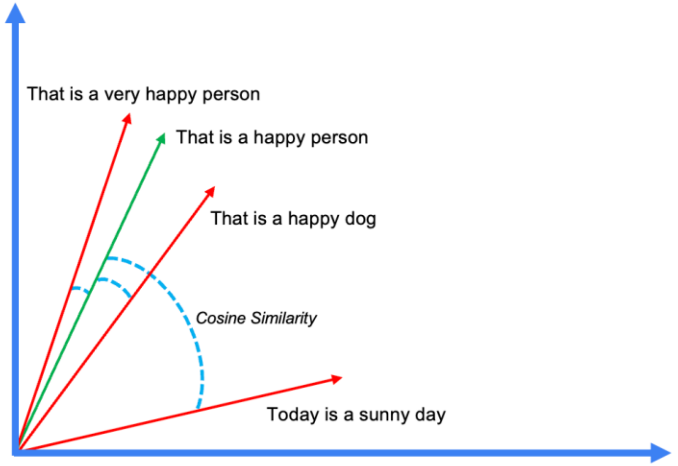

Acknowledgement: Slides taken from Sam Partee, Applied AI

Vector Similarity Search

- 3 semantic vectors = **Search Space**
  - “today is a sunny day”
  - “that is a very happy person”
  - “that is a very happy dog”

- 1 Semantic vector = **Query**
  - “That is a happy person”

Goal: Find most similar vector to the query



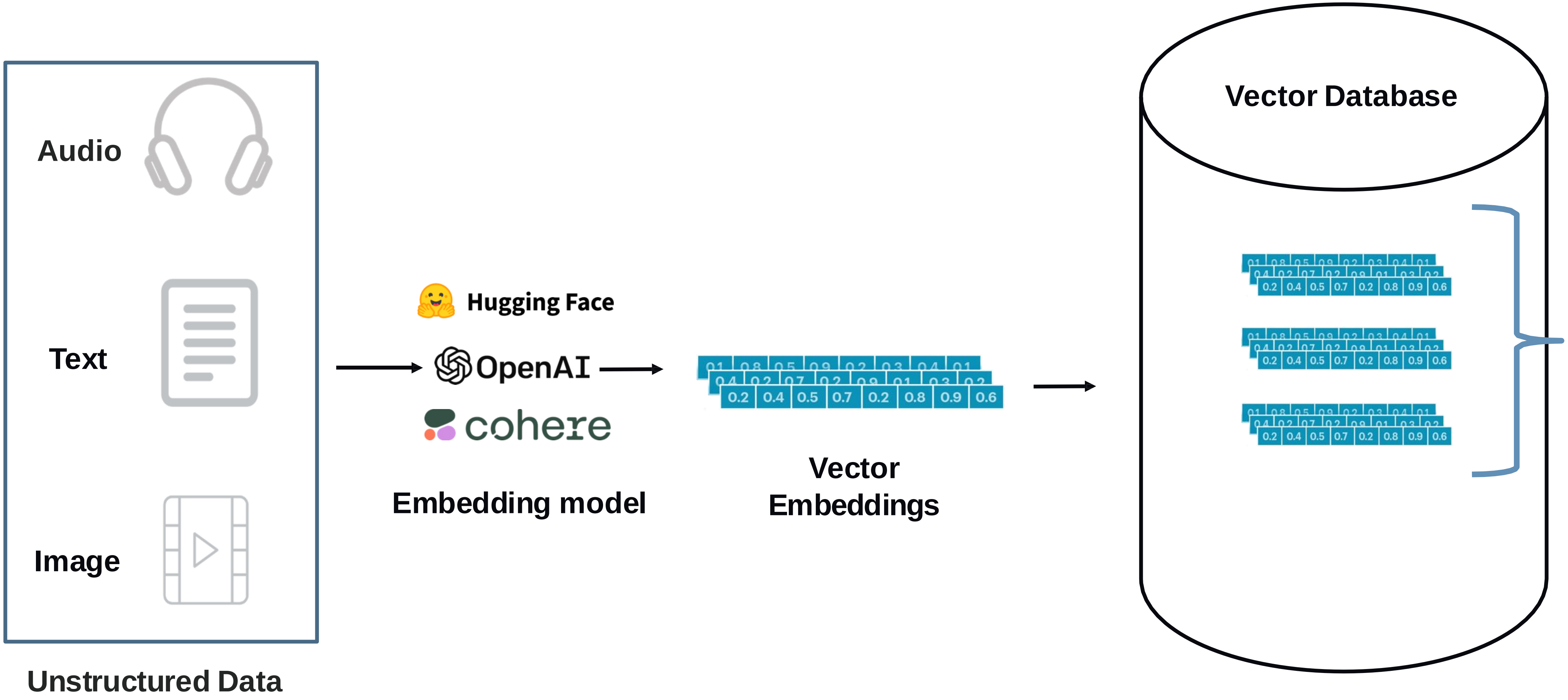
How? Calculate the distance (ex. Cosine Similarity)

That is a happy dog	0.695
That is a very happy person	0.943
Today is a sunny day	0.257

SCHOOL OF COMPUTING
UNIVERSITY OF UTAH

Acknowledgement: Slides taken from Sam Partee, Applied AI

Vector database



Acknowledgement: Slides taken from Sam Partee, Applied AI

Nearest neighbor search

Recall: Given a set $X \in \mathbb{R}^d$ of size n.

Goal: For query $q \in \mathbb{R}^d$, find nearest neighbor $x^* = NN_X(q) = \arg\min_{x \in X} ||x - q||$.

Two phases:

1. Build data structure (hopefully not too much larger than $|X|$)
2. Answer queries for $q \in \mathbb{R}^d$.

Nearest neighbor search

We introduce a new strategy today: Greedy Graph Search

1. Build a sparse graph $G = (X, E)$ on X, with $E(x)$ including at least is near neighbors
2. On query q, start with (*any*, *random*?) node $x \in X$, and see if any neighbors $x' \in E(x)$ are closer. If so, recurse on x'.
   (More robust (and useful) to maintain k closest point.)
   Terminate when no improvement is possible.

If each point $x \in X$ has at most m neighbors,
and each path is at most h hops,
then this approach has

- $O(nm)$ space
- $O(mh)$ query time.

Nearest neighbor search

Hierarchical Navigable Small World Graphs (HNSW)

Includes Neighborhood graph, undirected

Consider $L \leq \log n$ levels of edge length in graph

Each $x \in X$, for each level $\ell \in L$, choose $\approx$ closest point

Approximate by building points $x_1, x_2, \ldots \in X$, where $X_i = \{x_1, x_2, \ldots, x_i\}$

When building $E(x_i)$ find closest K among points in X_{i-1}

Add reverse nearest neighbor edges (undirected)

Make sure includes K' nearest neighbors.

Start search from one of first $x_1 \ldots, x_s$ for small s (with long edges).

RAG implementation

FAISS

Facebook AI Similarity Search

mostly for $x^* = \arg\min_{x \in X} ||x - q||$

Combination of 2 ideas

1. quantized index
2. GPU acceleration

Quantized index

database vector as $x \approx Q(x) = Q_1(x) + Q_2(x) + \ldots + Q_m(x)$

- where $Q_j : \mathbb{R}^d \to C_j$
  so C_j is a *codebook* of size k (k points in $\mathbb{R}^d$)
  e.g., k centers of k-means clustering
- C_j has more detail than C_{j-1}
  C_j has more ``weight" than C_{j+1} – measures larger distances
- m is small 2, 3, ... 6?

Roughly we want

- the same number of points $X_j \subset X$ which quantize to the $c_j \in C_1$
- the maximum distance $\max_{x \in X_j} \|c_j - x\|$ similar for each c_j.
  (Should be feasible if doubling dimension bounded, and measure fairly uniform)

Quantized index

Then quantize each X_j with next another k codewords with that set recursively down to $C_2, C_3, ...C_m$.

each distance stored with limited precision (over limited range) --> saves space

On search $q \in \mathbb{R}^d$:

- find $c^* = \arg\min_{c \in C_1} ||q - c||$
- recurse on X_{j*} and its quantization C_{j*}
- data adaptive, very-wide hierarchical index

More efficient and robust with maintaining top-K

GPU acceleration

GPU acceleration

The problem with very-wide architecture is that

... find $c^* = \arg\min_{c \in C_1} ||q - c||$

requires a NN search!

GPUs are fast parallel processes

can min of k operations at once

solves N on size-k codebooks efficient

RAG and Pinecone

- Pinecone is a Billion-dollar company
- Based on graph-based similarity search
- Build to deliver RAG for companies

LLM + Vector DB Use Cases

Because large was not large enough

Vector database for LLMs

Context Retrieval

- Search for relevant sources of text from the “knowledge base”
- Provide as “context” to LLM

LLM “Memory”

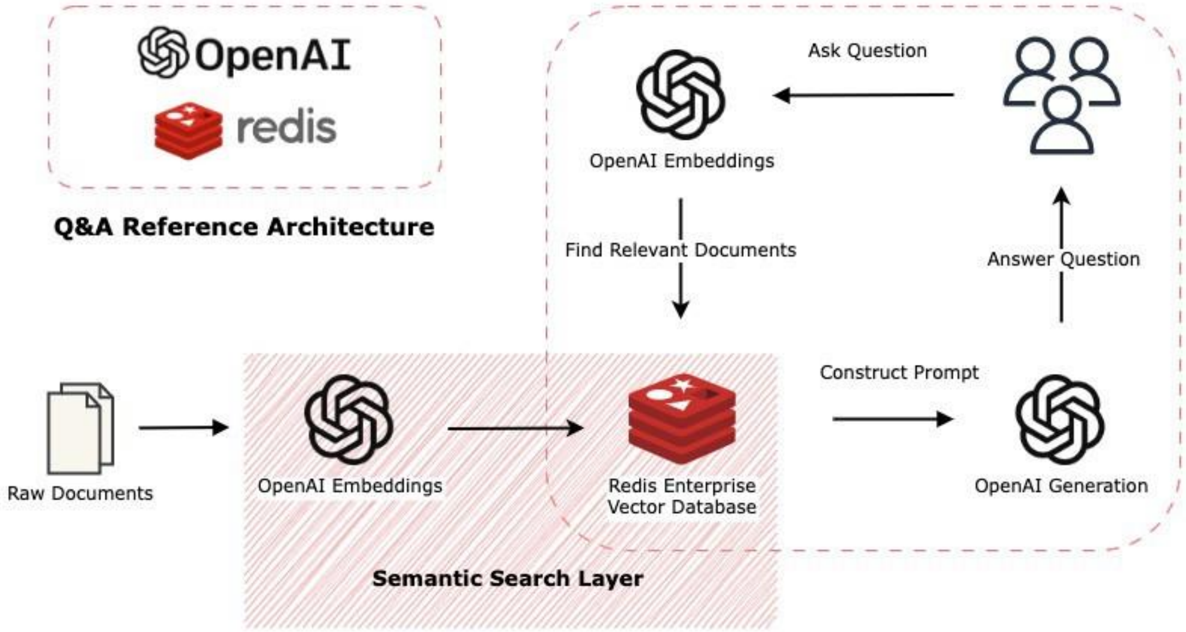
- Persist embedded conversation history
- Search for relevant conversation pieces as context for LLM

LLM Cache

- Search for semantically similar LLM prompts (inputs)
- Return cached responses

Acknowledgement: Slides taken from Sam Partee, Applied AI

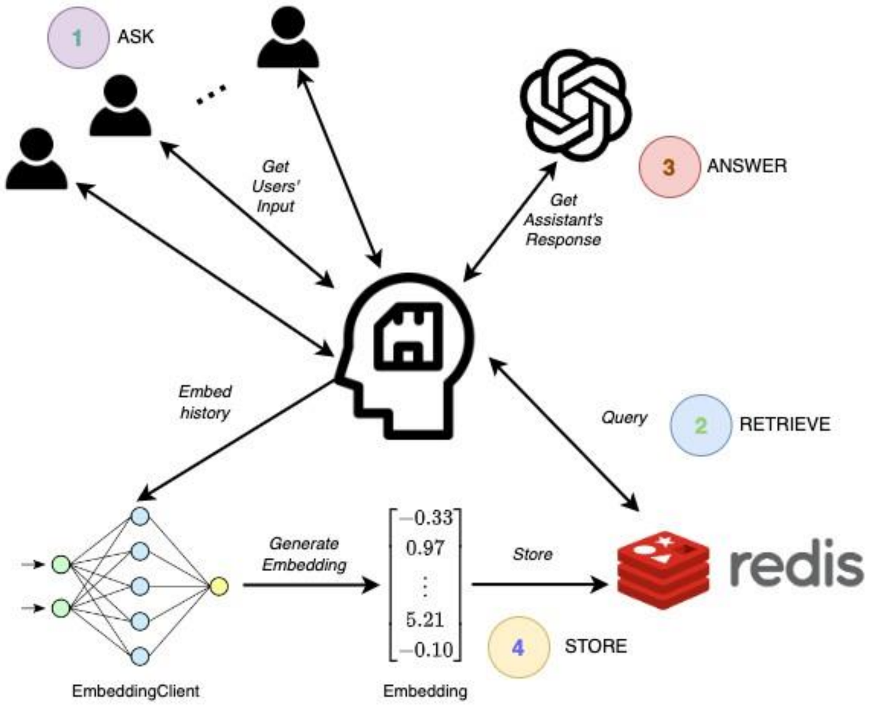
Context retrieval



- Description
  - Vector database is used as an external knowledge base for the large language model.
  - Queries are used to detect similar information (context) within the knowledge base
- Benefits
  - **Cheaper and faster** than fine-tuning
  - **Real-time updates** to knowledge base
  - **Sensitive data** doesn't need to be used in model training or fine tuning
- Use Cases
  - Document discovery and analysis
  - Chatbots

Acknowledgement: Slides taken from Sam Partee, Applied AI

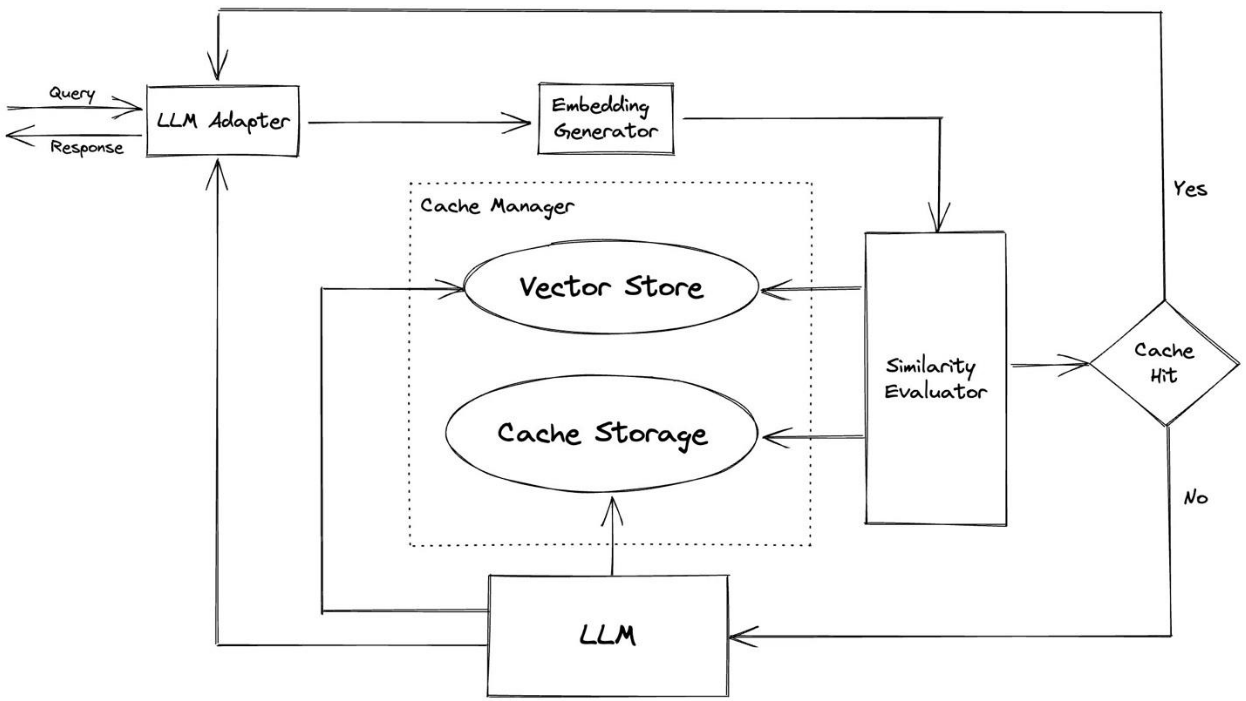
Long term memory for LLMs



- **Description**
  - Theoretically infinite, contextual memory that encompasses multiple simultaneous sessions
  - Retrieves only last K messages relevant to the current message in the entire history.
- **Benefits**
  - Provides **solution to context length limitations** of large language models
  - Capable of **addressing topic changes** in conversation without context overflow
- **Use Cases**
  - Chatbots
  - Information retrieval
  - Continuous Knowledge Gathering

Acknowledgement: Slides taken from Sam Partee, Applied AI

LLM query caching



- Description
  - Vector database used to cache similar queries and answers
  - Queries embedded and used as a cache lookup prior to LLM invocation
- Benefits
  - **Saves on computational and monetary cost** of calling LLM models.
  - Can **speed up applications** (LLMs are slow)
- Use Cases
  - Every single use case we've talked about that uses an LLM.

Acknowledgement: Slides taken from Sam Partee, Applied AI

Parting thoughts

- Vector databases are hot and underlie modern ML-based platforms
- There are still a bunch of open research questions regarding
  - Most efficient indexing technique for managing embeddings