ΜΥΕ003: Ανάκτηση Πληροφορίας

Διδάσκουσα: Ευαγγελία Πιτουρά

Σημασιολογική Ανάκτηση Πληροφορίας. Learning to Rank. Vector Search. RAG.

Ακαδημαϊκό Έτος 2024-2025

Περίληψη

- Διαβάθμιση βασισμένη σε tf-idf

  ορισμός, ανάκτηση

  διανυσματική αναπαράσταση

- Embeddings (ενσωματώσεις)
  - μαθαίνουμε τη διανυσματική αναπαράσταση
  - word2vec embeddings
    - Βασική αρχιτεκτονική: CBOW, Skipgram
      - sentence, paragraph, document embeddings

Περιεχόμενα

- Σημασιολογική (Διανυσματική Ανάκτηση)
- Learning to Rank
- Retrieval-Augmented Generation (RAG)

Semantic (vector) Search

Διανυσματική Ανάκτηση

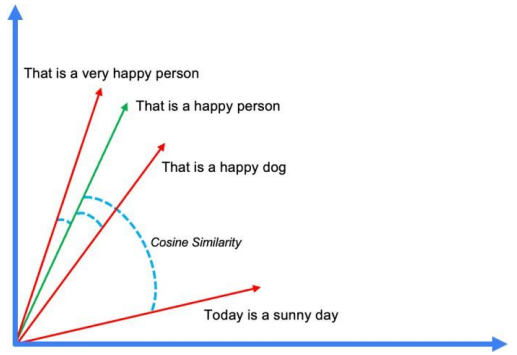
1. Αναπαράσταση κάθε εγγράφου ως ένα διάνυσμα
2. Αναπαράσταση του ερωτήματος ως ένα διάνυσμα
3. Υπολογισμός της (cosine) ομοιότητας για κάθε ζεύγος ερωτήματος, εγγράφου
4. Διάταξη των εγγράφων με βάση την ομοιότητα

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"

Vectors generated by some embedding method

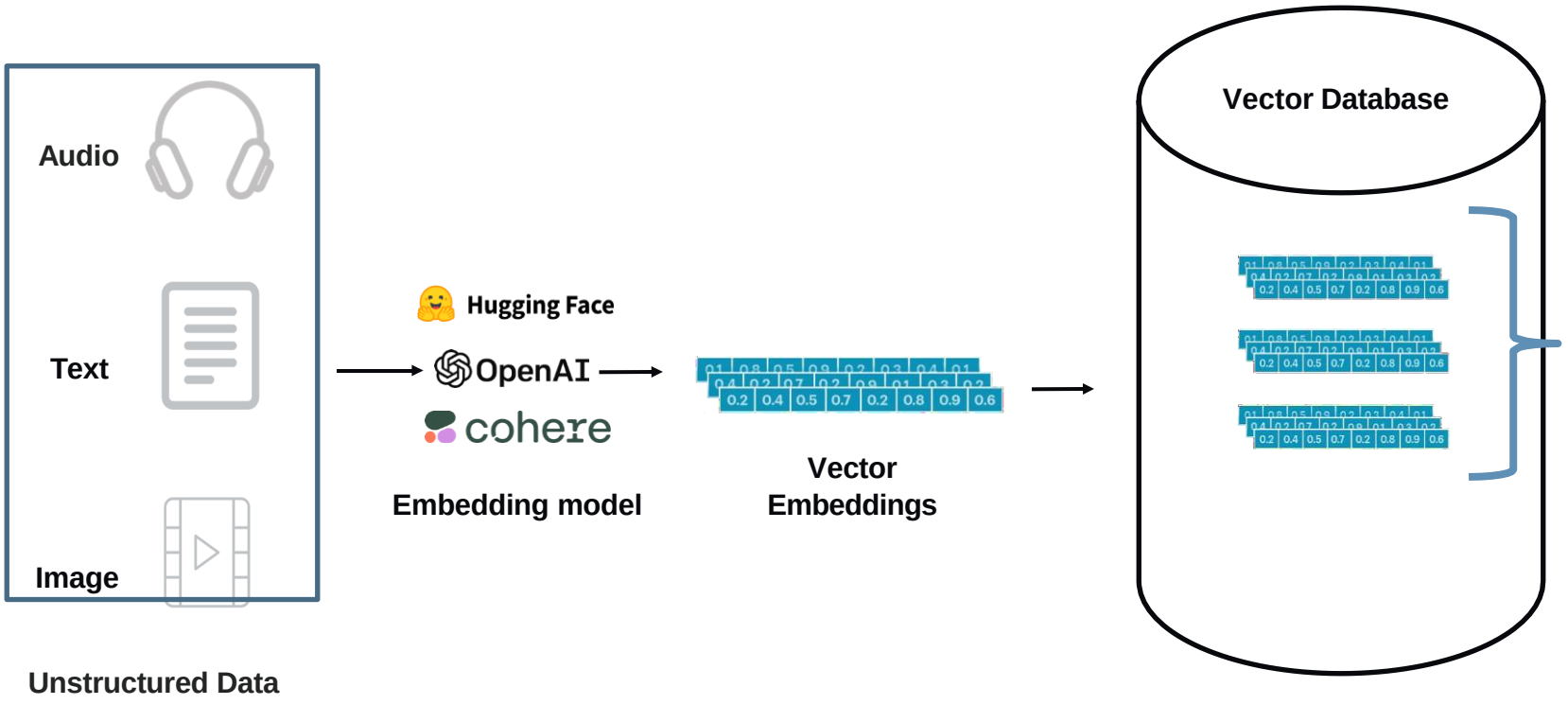
Goal: Find most similar vector to the query



How? Calculate the distance (ex. Cosine Similarity)

Acknowledgement: Slides taken from Sam Partee, Applied AI

Vector database



Acknowledgement: Slides taken from Sam Partee, Applied AI

Slides by Prashant Pandey

Semantic or vector search

Semantic (vector) search Given a dense vector query q and a vector database of document vectors return the *k*-most similar (e.g., using cosine similarity) documents to *q*

K-nearest neighborhood search
Approximate K-nearest neighborhood search

- Baseline approach

calculate the distance between *q* and all documents

How to do this efficiently?

Classic (symbolic) search: dictionary and inverted indexes

Semantic or vector search

Index-based approaches

- Hashing-based
- Tree-based
- Quantization-based
- Proximity graph (PG)-based

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

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)

Graph-based vector databases

- Utilize *graph structures* (e.g., k-NN graphs, Hierarchical Navigable Small-World (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

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)

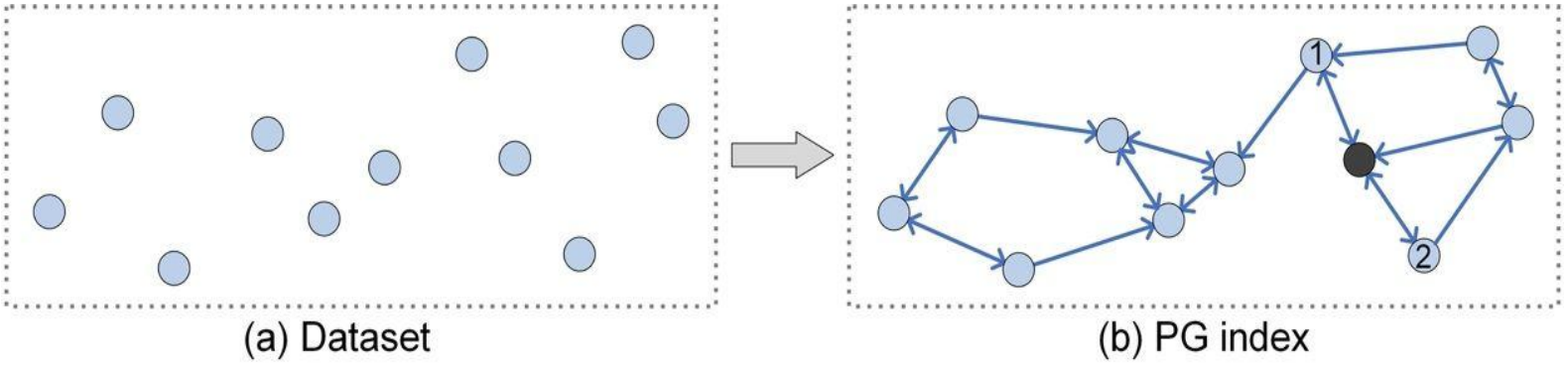
Cloud-native vector databases

- 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)

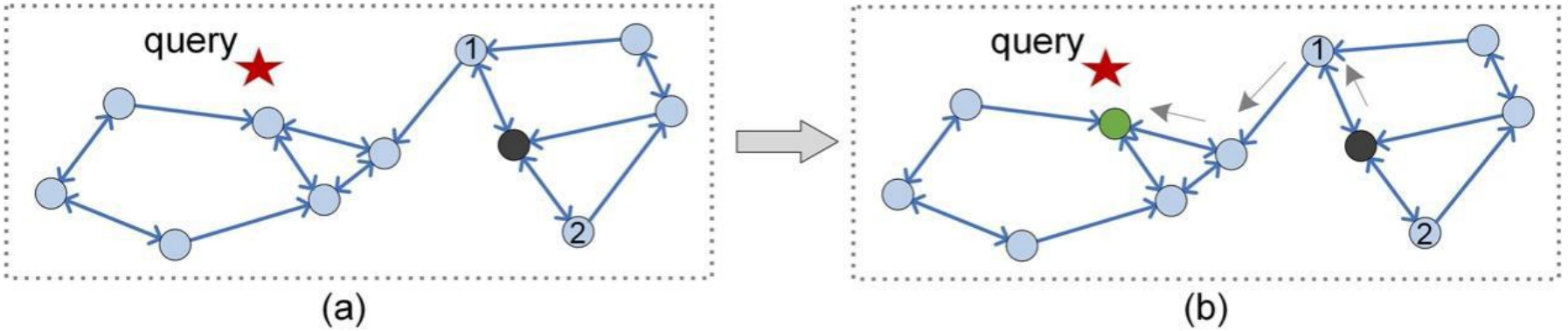
Proximity Graph (PG) based



link every vertex to its k nearest neighbors in the dataset

Vertex: document
Edge x -> y if y is a k-neighbor of x

Proximity Graph (PG) based



Select a seed vertex (the black vertex)

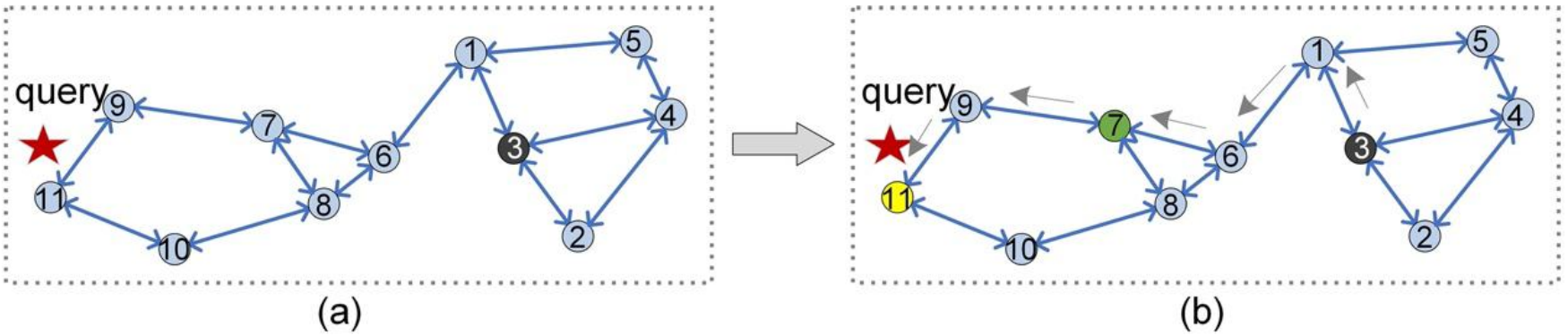
randomly sampled or obtained by an additional index such as tree, is selected as the result vertex r

Conduct routing from the seed vertex

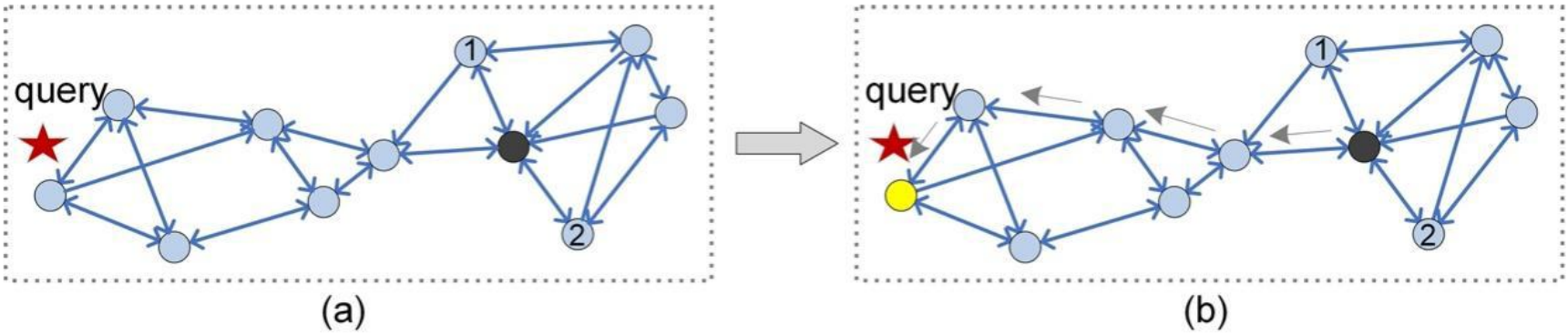
if d(n, q) < d(r, q), where n is one of the neighbors of r, n replaces r

Repeat

Proximity Graph (PG) based



Add reverse edge



Increase k

Many more optimizations

Semantic or neural search

Classic (symbolic) search: dictionary and inverted indexes

Semantic/neural search: map documents to embeddings (low dimensional dense representations)

Vector index
+ resilient to noise, scale
- needs lots of data to train, lack of explainability

Neural (semantic) search in Lucene (solr)

Given a dense vector q that models the query (information need) calculate the distance (Euclidean, cosine, dot product, etc.) between q and every vector d that represents a document

Nearest Neighborhood Search

Proximity Neighborhood Graph

- Vertices (documents) are mapped to nodes
- Edges between “similar” vertices

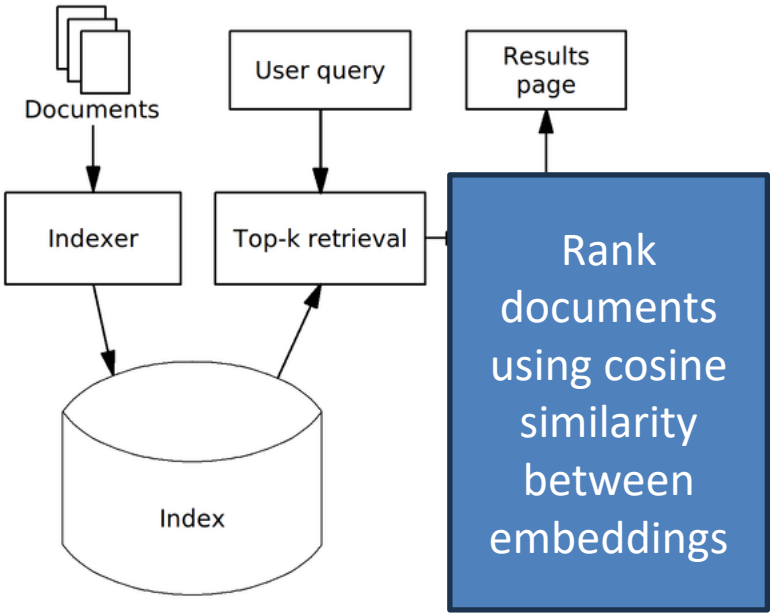
Navigable small world graph

https://sease.io/2022/01/apache-solr-neural-search.html
https://sease.io/2023/01/apache-solr-neural-search-tutorial.html

Combine tf-idf and vector similarity

Use tf-idf as a filter

- Get the top-*k* documents (e.g., document that have at least one of the query terms)
- Use the ranking models to create the final ranking of the top-*k* documents



Περιεχόμενα

- Σημασιολογική (Διανυσματική Ανάκτηση)
- Learning to Rank
- Retrieval-Augmented Generation (RAG)
- Lucene Project

Learning to rank

Learning to rank

Retrieval models so far

Classical: utilize exact matching signals to design a relevance scoring function (e.g., term frequency, document length, and inverse document frequency) to rank documents

Semantic: *Learn representations* and use *similarity* to rank documents

Supervised learning for ranking

Ground truth (labeled data) regarding the relevance of documents to queries

- Manually labeled
- Historical data: Clickthrough data

Used them as training data to learn a ranking model

Learning to rank

- Input:
  Document features (tf, embeddings, *k*-grams, etc)
  Query features

- Learn a *ranking function f (a ranking model)*

  - By solving a minimization problem with respect to a *loss function* which is a measure of accuracy with respect to the training data

How do we specify the ranking?

Learning to rank

1. Point-wise approach

- Training data (ground truth)

  For each query-document pair (q, d) there is a numerical, or ordinal score

- The learning-to-rank problem can be approximated by *a regression problem*:

  Given a (q, d) pair, *predict its score*

- Existing supervised machine learning algorithms can be used for this purpose.

Learning to rank

2. Pair-wise approach

- Training data

  For each query q, a *pair of documents* (d_1, d_2), where d_1 is better than d_2

- The learning-to-rank problem can be seen as *a binary classification problem*:

  Given a (d_1, d_2), pair output 1 of d_1 is better and 0 otherwise

- Existing supervised machine learning algorithms such as probabilistic classifiers (e.g., logistic regression) can be used for this purpose.

Learning to rank

3. List-wise approach

- Training data not a single, or a couple of documents but an ordered list.
  For example:
  For each query q,
  a list of documents d_i associated with scores/judgements r_i

- Learning-to-rank

$$L_{LN}\big(r(q), \hat{r}(q)\big)$$

minimize the sum of the differences between the predicted $\hat{r}(q)$ and the actual score $r(q)$.

Learning to Rank

Learning to rank models

apply supervised machine learning techniques to solve ranking problems using hand-crafted, manually-engineered features

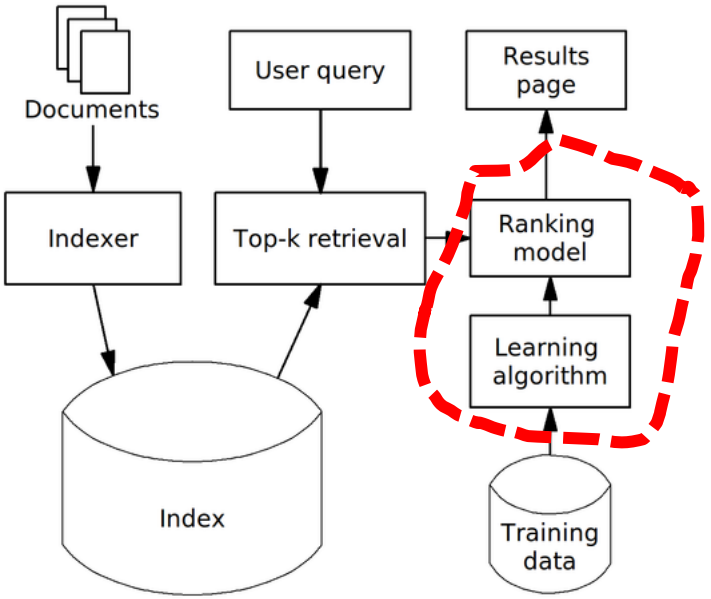
Neural retrieval models

use as input the embeddings of q and d are usually trained in an *end-to-end* manner with relevance labels.

Learning to rank in IR

Use tf-idf as a filter

- Get the top-*k* documents (e.g., document that have at least one of the query terms)
- Use the ranking models to create the final ranking of the top-*k* documents



- Input: Documents, query
- Learn some *(partial) order of the documents* based on the *relevance of each document for the query*

There may be multiple re-ranker modules

Retrieval Augmented Generation (RAG)

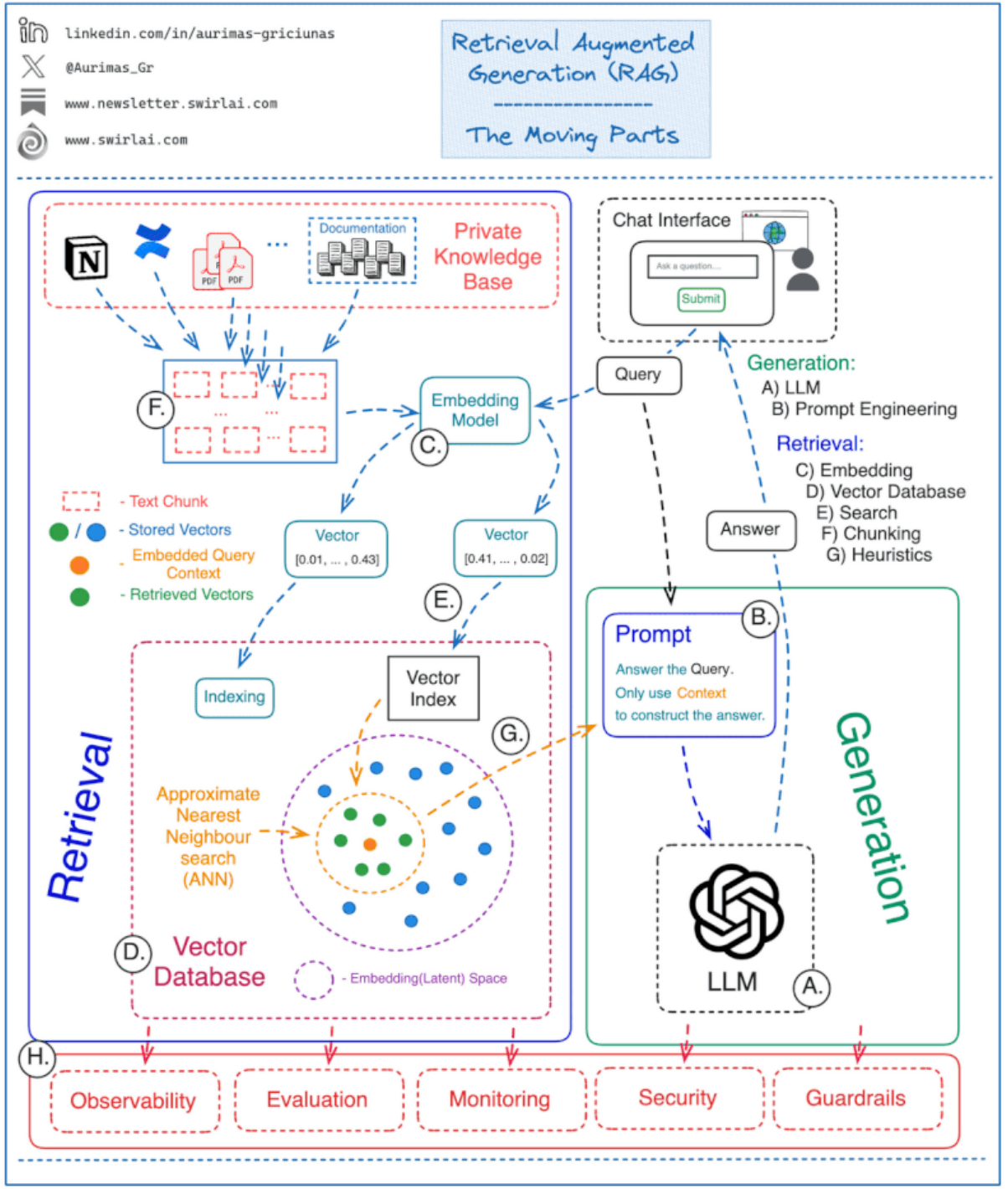
Retrieval Augmented Generation

- Large Language Models (e.g., ChatGPT) suffer from hallucination.
- Also lack current, or domain-specific information

- Retrieval Augmented Generation (RAG) is a framework designed to make LLMs more reliable by retrieving documents relevant to a user query.

RAG

Retrieval-Augmented Generation



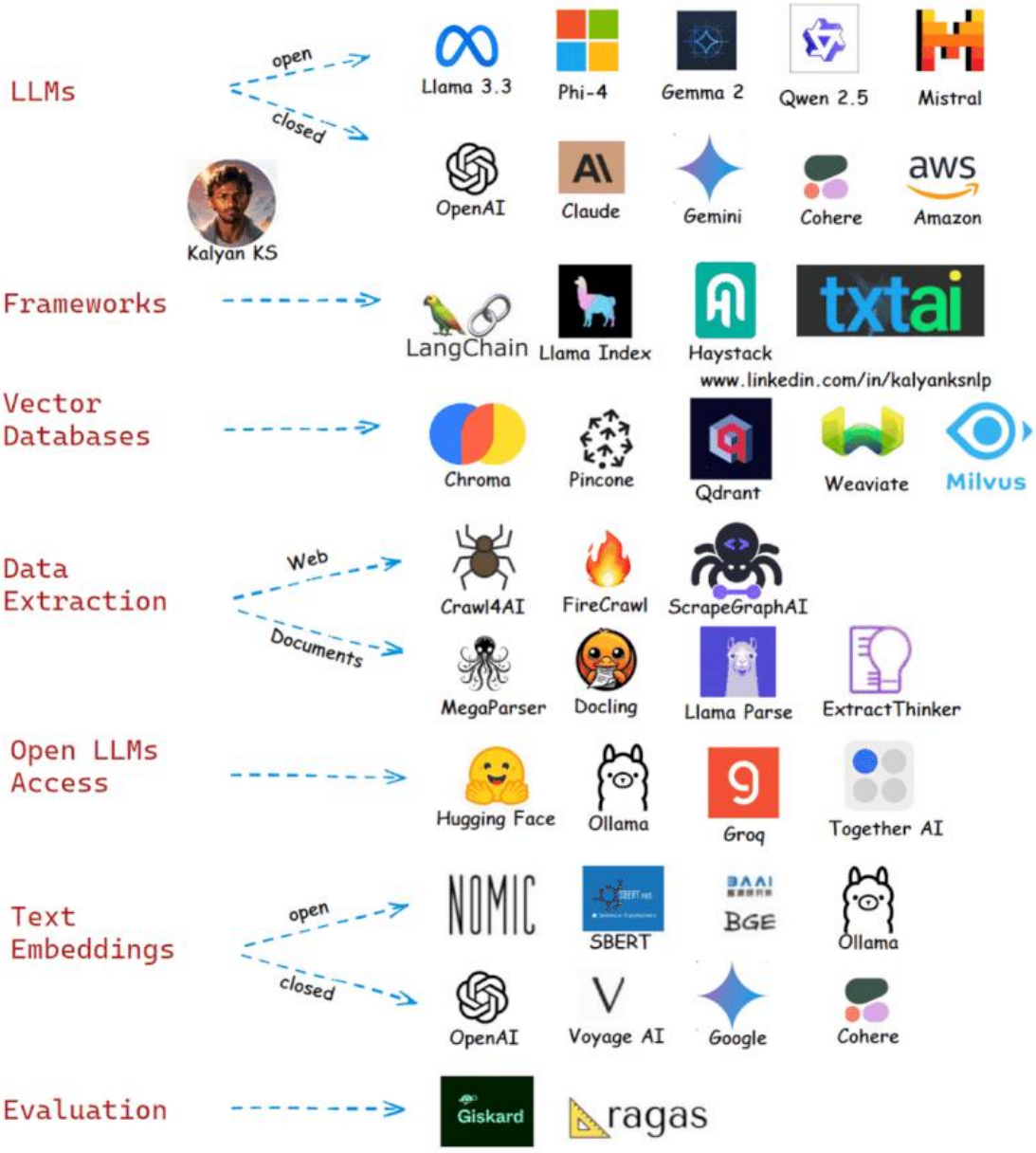
Create external data
Data outside of the LLM original training data
From multiple data sources, such as a APIs, databases, or document repositories.
Embedding models, converts data into numerical representations and stores it in a vector database

Chunking algorithms: break down documents into semantically cohesive chunks

Retrieve relevant information
The user query is converted to a vector representation and matched with the vector databases.

Augment the LLM prompt
Augments the user input (or prompts) by adding the relevant retrieved data (text) in context. This step uses prompt engineering techniques to communicate effectively with the LLM.

RAG Developer's Stack



LLM + Vector DB Use Cases

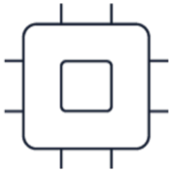
Vector database for LLMs

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

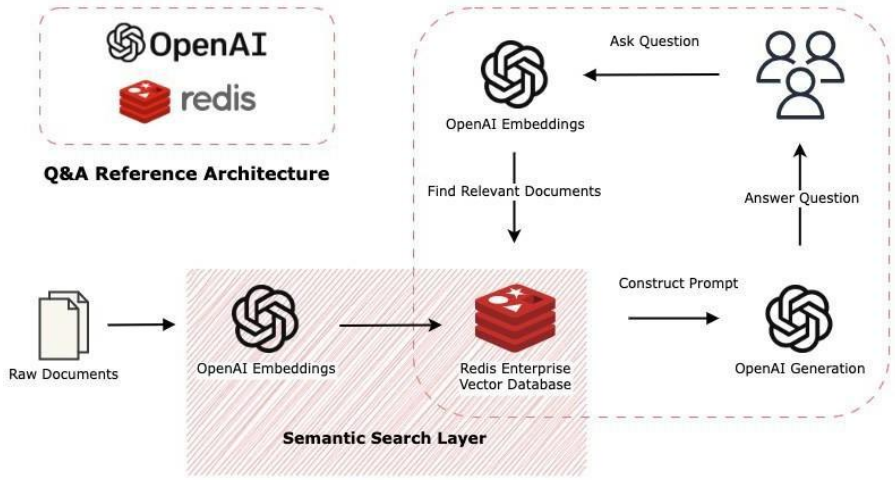


LLM Cache

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

Acknowledgement: Slides taken from Sam Partee, Applied AI

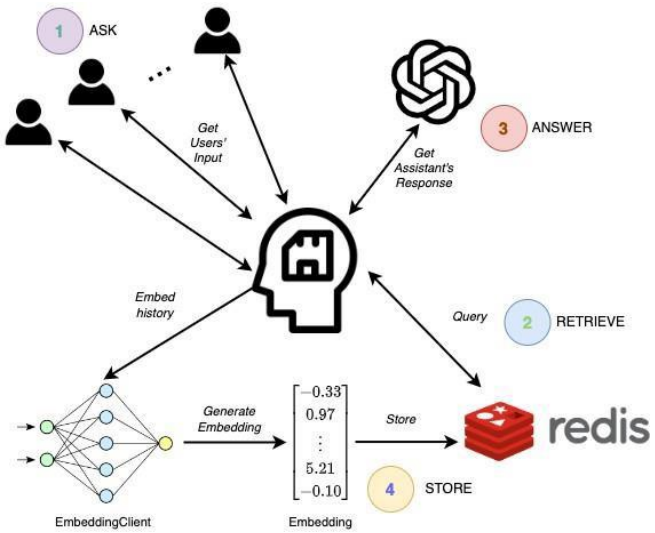
Context retrieval (RAG)



- 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

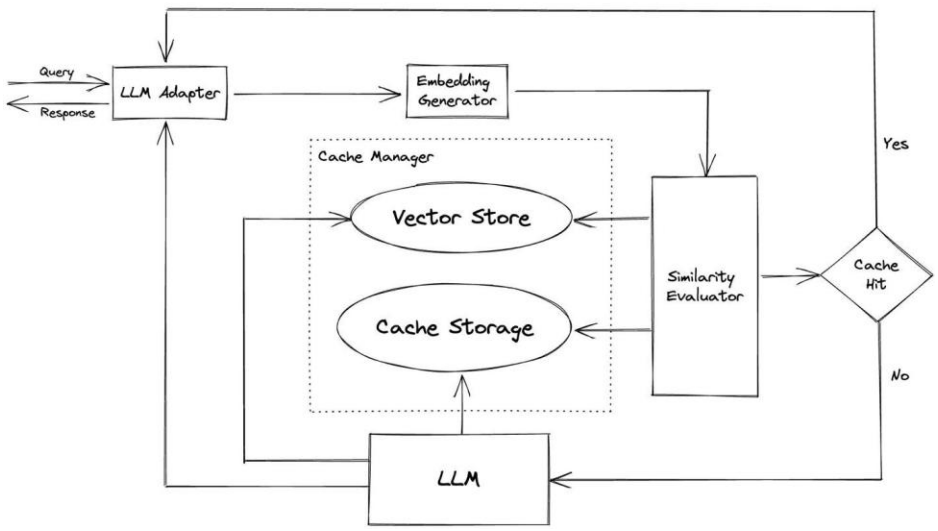
Long term memory for LLMs



- **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

End of lecture

Χρησιμοποιήθηκε υλικό από

- CS276: Information Retrieval and Web Search, Christopher Manning and Pandu Nayak, Lecture 14: Distributed Word Representations for Information Retrieval
- https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Μια περιγραφή του skipgram:
Chris McCormick
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
Δείτε και το
https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/