[CHAT-9.0] RETRIEVAL-AUGMENTED GENERATION

LLM taxonomy

- **Text generation** models: generate new text given another text, called the **prompt**. Other names: causal, autoregressive.
  Example: Google's Gemini 2.5 Flash, OpenAi's GPT-4o mini.
  - Code models.
  - General-purpose models.
  - **Reasoning** models.
- **Embedding models**: generate vector representations. Examples: Google's text-embedding-004, OpenAI's text-embedding-3-small.

What is an embedding?

- An **embedding** is a representation of a piece of information, such as a word, a sentence or an image, as a vector in a space of a given dimension. Typical **embedding dimensions**, for the embedding models that you can manage in your computer, are 384, 512, 768 and 1,024. Nevertheless, the top performing LLMs work with higher embedding dimensions.
- For an embedding to be useful, “similar” pieces of information are represented by vectors that are close in a geometric sense. For instance, in a word embedding, words with similar meanings, such as ‘nice’ and ‘beautiful’, will be represented by close vectors. Unrelated words, such as ‘computer’ and ‘dog’, will be represented by non-close vectors.
- When we use an ML model to create embedding vectors associated to images or texts, we say that we are “encoding” them. In particular, the large language models used for that purpose are called **encoders**. The most famous of these encoders is Google’s BERT.

Applications of embeddings

- Clustering.
- Prediction.
- Recommendation. This is explained later in this note, and illustrated in the example that follows.
- Outlier detection.
- Text generation LLMs.
- Semantic search.
- Retrieval-augmented generation.

Vector databases

- After encoding a collection of documentes as vectors, how can we store these vectors in such a way that they can be efficiently retrieved?
- The response (so far) of the industry to this question is the **vector database**:
  - Free databases like **ChromaDB**.
  - Commercial databases like **Pinecone**.
  - Old databases adapted to vectors, like **PostgreSQL**.

Three approaches to improve your chat model

- Retrieval-augmented generation (RAG).
- Fine-tuning.
- Prompt engineering.

Retrieval-augmented generation (RAG)

- Pros
  - Up-to-date.
  - Domain specific.
- Cons
  - Performance.
  - Processing.

Fine-tuning

- Pros
  - Deep domain expertise.
  - Faster.
- Cons
  - Training complexity.
  - Computational cost.
  - Maintenance.
  - Catastrophic forgetting.

Prompt engineering

- Pros
  - No infrastructure.
  - Immediate results.
- Cons
  - Trial and error.
  - Limited to existing knowledge.