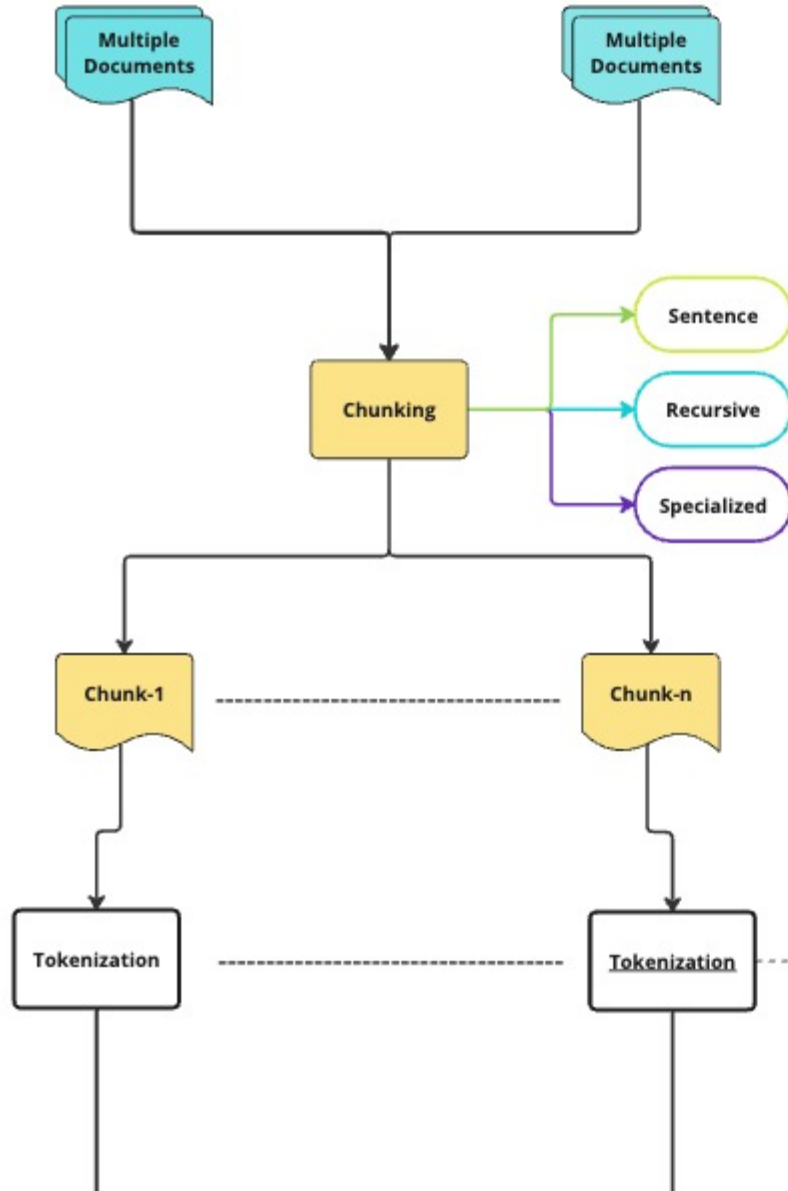


Stage 1 : Data Preparation



Choosing Chunk Size

- LLM Context window:
 - Limit on how much data you can input to an LLM
 - Top-K Retrieved Chunks
- High context length = quadratic increase in time & memory
 - Due to transformer model's self attention mechanism

The diagram shows a central box labeled 'LLM Context Window' with arrows pointing to various components: 'Prompt', 'System', 'User', 'Assistant', 'Tools', 'Plugins', 'Memory', and 'History'.

KDB.AI
Chunking Best Practices for RAG Applications
Watch Now
YouTube · Updated 9 hours ago

LLMs + TEXT CHUNING
Watch Now
YouTube · Updated 9 hours ago



Transformers
ECCV 2020

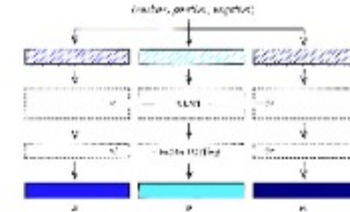
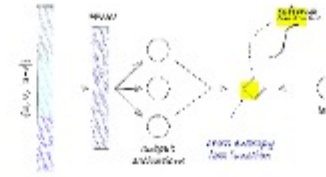
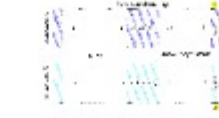
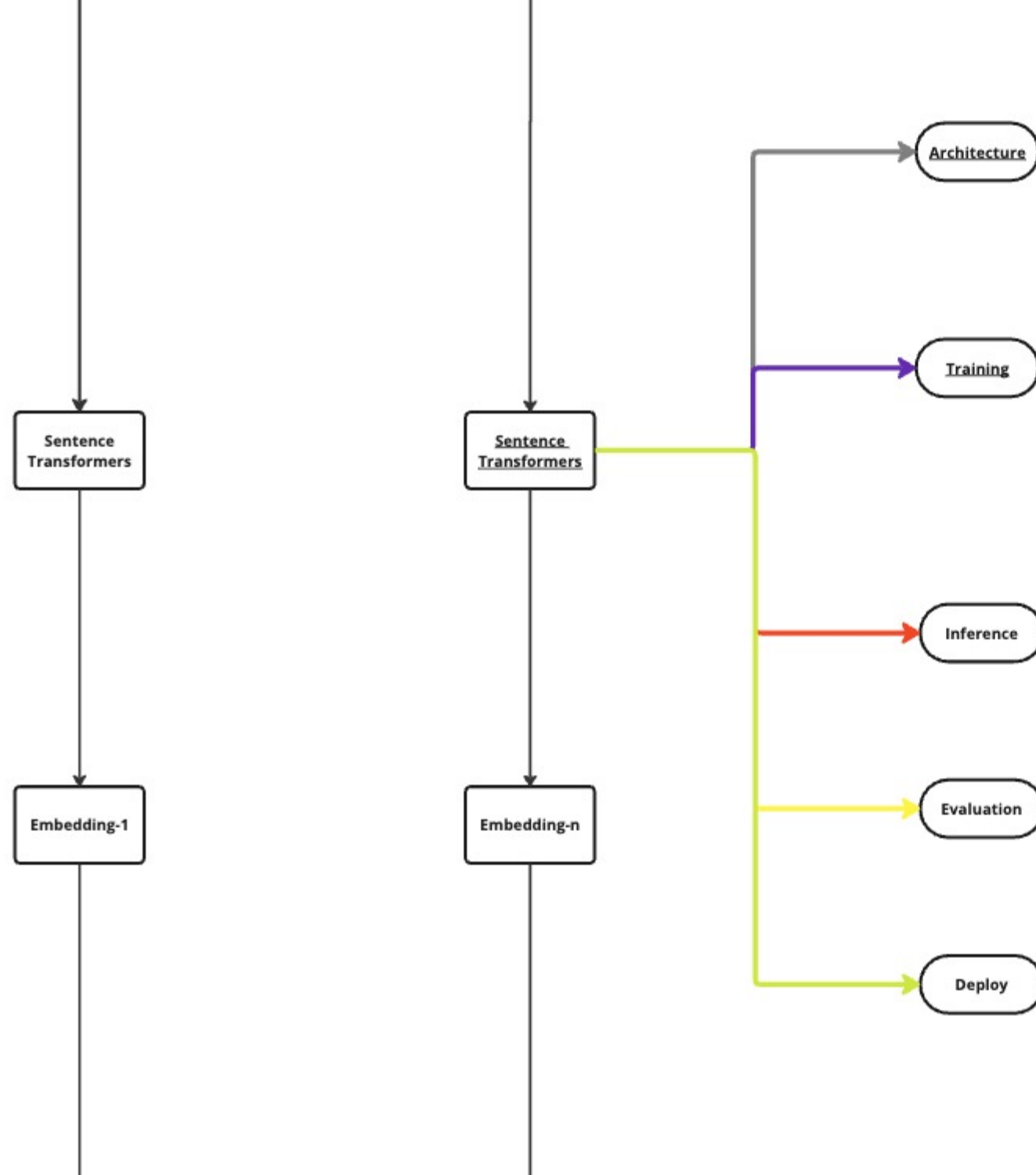
huggingface.co

Summary of the tokenizers

We're on a journey to advance and democratize artificial intelligence through open source and open science.

Building RAG-based LLM Applications for Production
In this guide, we will learn how to develop and productionize a retrieval augmented generation (RAG) based LLM application, with a focus on scale and evaluation.

Stage 2 : Embedding



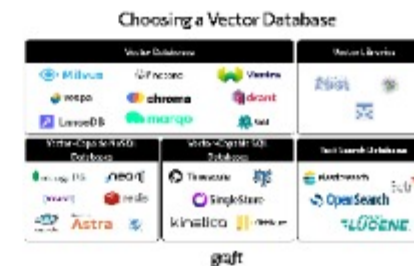
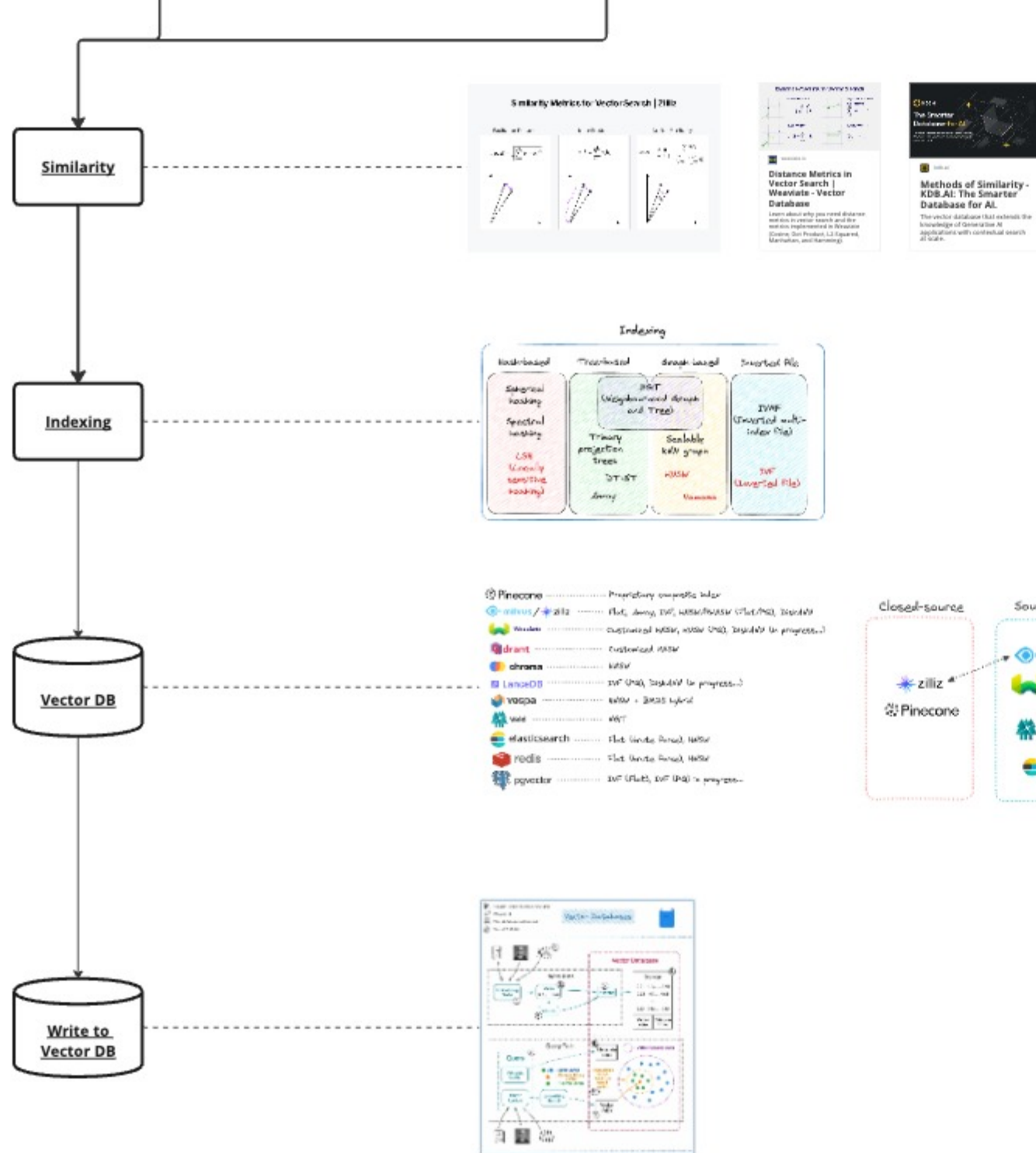
Pretrained Models - Sentence-Transformers documentation

We provide various pre-trained models. Using these models is easy: All models are hosted on the Huggingface Model Hub. The following table provides an overview of (selected) models. They have been extensively evaluated for their quality to embed sentences...

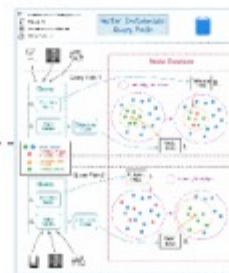
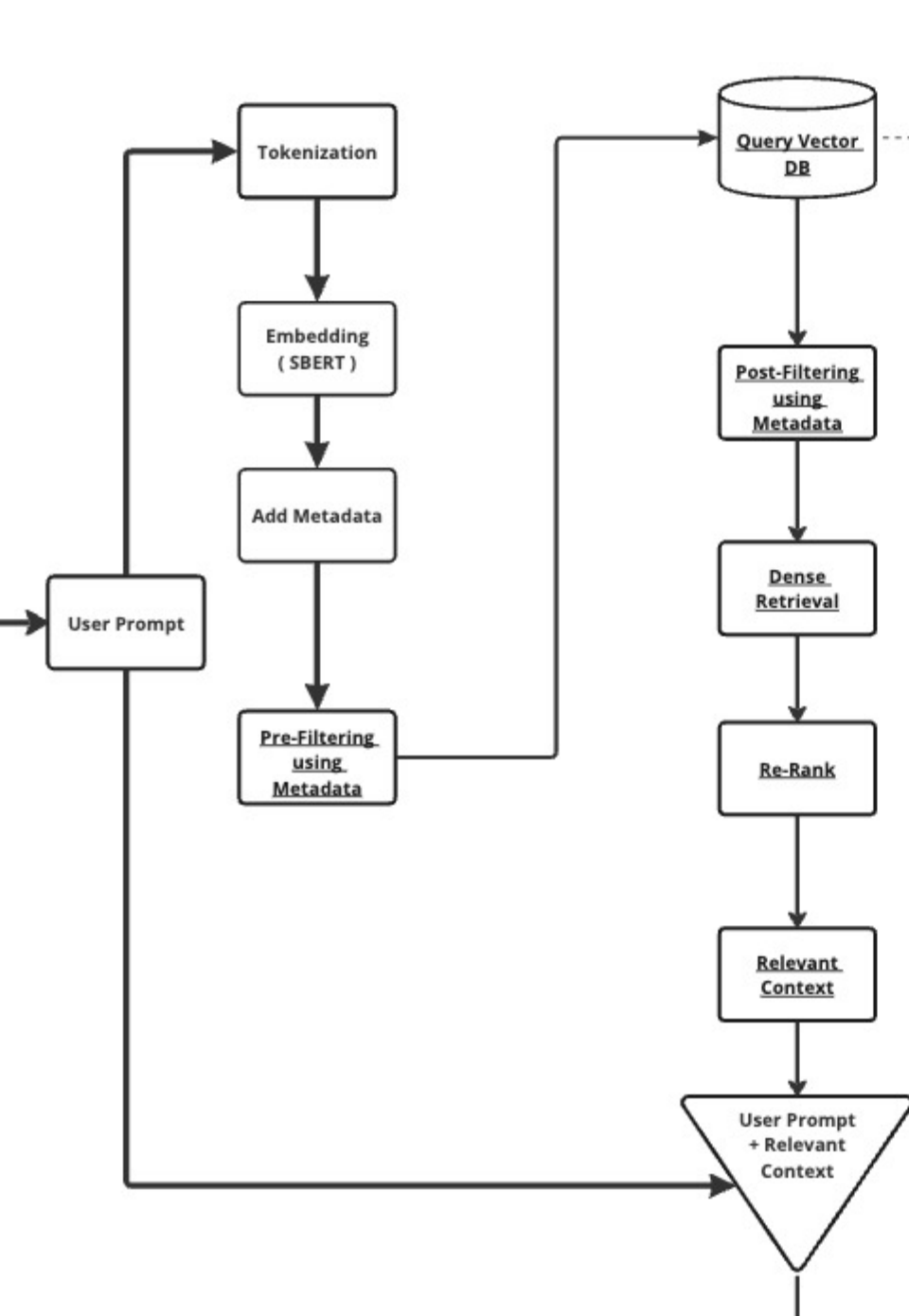
Evaluating RAG Part I: How to Evaluate Document Retrieval

A guide to the evaluation of components in retrieval augmented generation.

Stage 3 : Add / Update Record in Vector DB



Stage 4 : Query Processing + Semantic Search



Semantic Search - Sentence-Transformers documentation

Semantic search seeks to improve search accuracy by understanding the content of the search query. In contrast to traditional search engines which only find documents based on lexical matches, semantic search can also find synonyms. The idea behind semantic search is to use word embeddings to represent the meaning of words and phrases. This allows the search engine to find documents that are semantically similar to the query, even if they don't contain the exact same words.

util.semantic_search

Instead of implementing semantic search by yourself, you can use the `util.semantic_search` function.

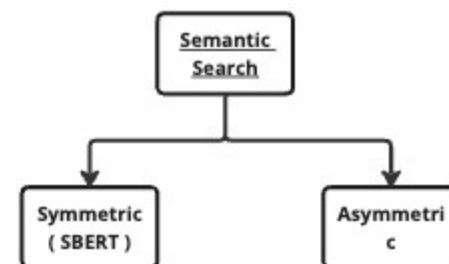
The function accepts the following parameters:

```
from sentence_transformers.util import semantic_search

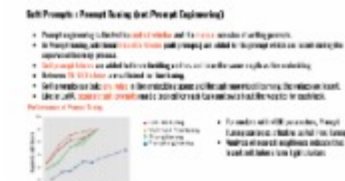
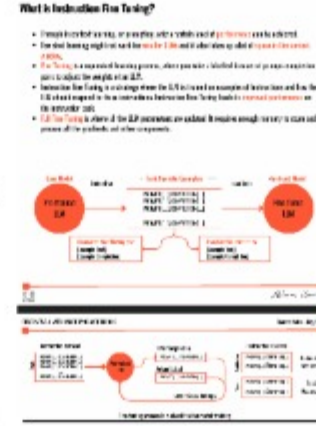
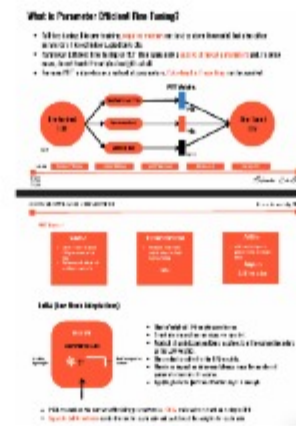
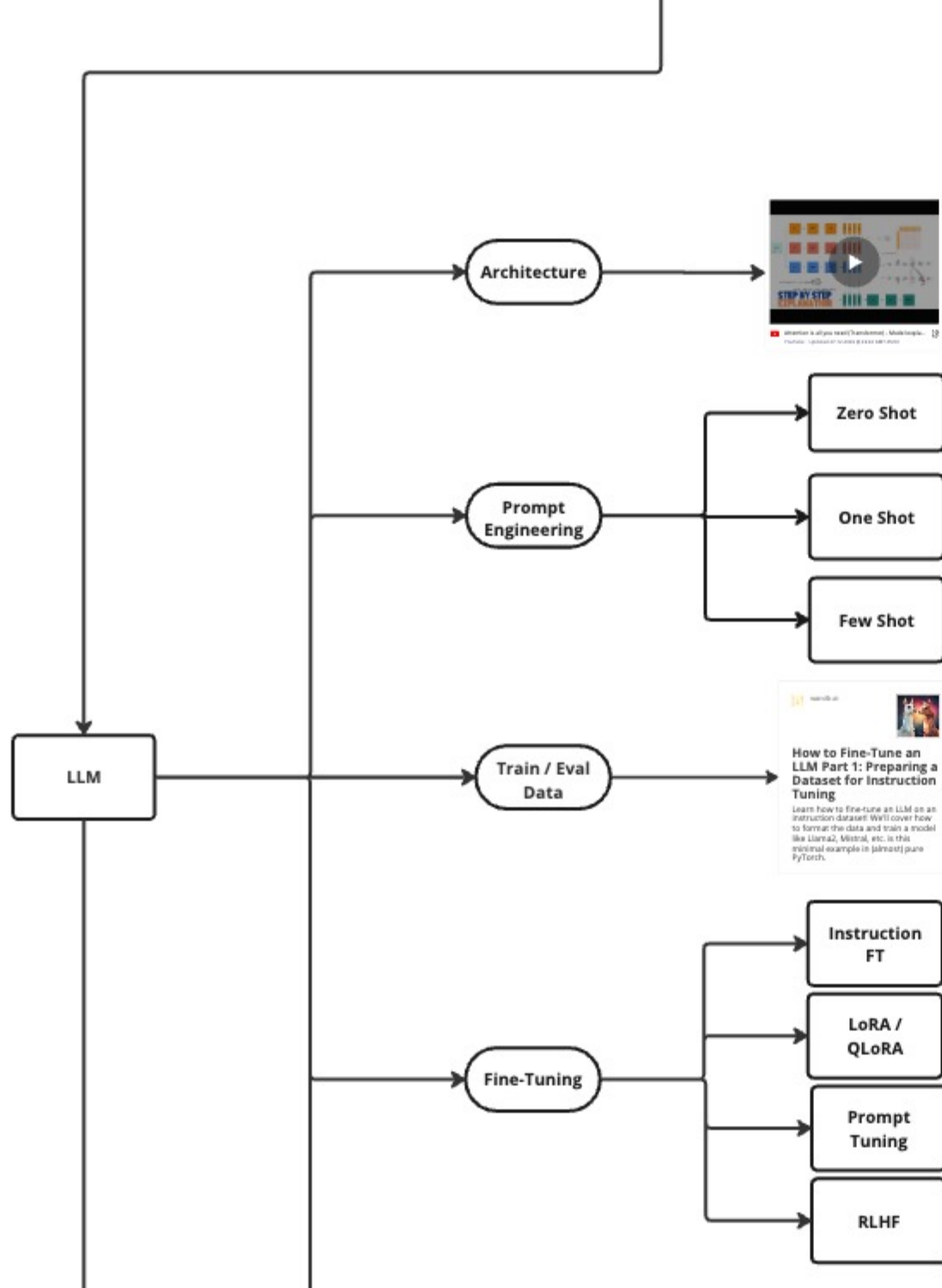
query_embeddings: torch.Tensor, corpus_embeddings: torch.Tensor, query_chunk_size: int = 100, corpus_chunk_size: int = 100000, top_k: int = 50, score_function: typing.Callable[[torch.Tensor, torch.Tensor], torch.Tensor] = <function cos_sim>
```

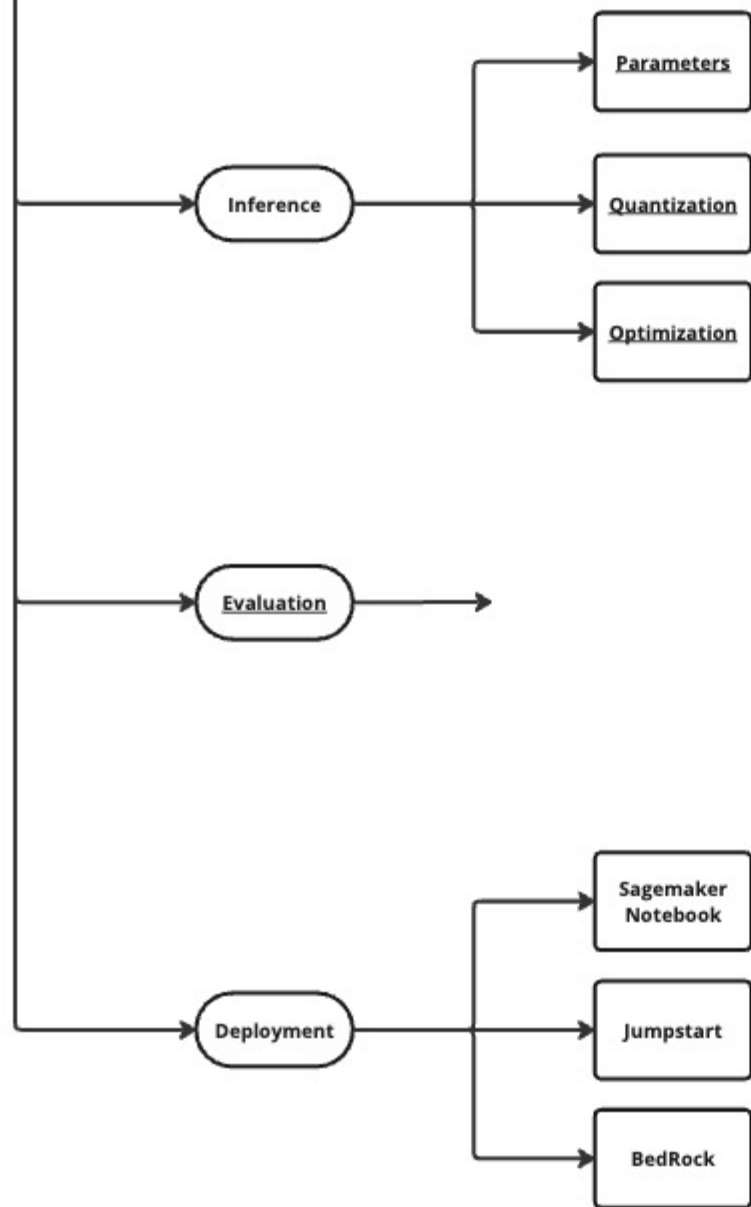
This function performs a cosine similarity search between a list of query embeddings and a list of corpus embeddings. It can be used for information retrieval / Semantic Search for corpora up to about 1 Million entries.

- Parameters:
- `query_embeddings` - A 2 dimensional tensor with the query embeddings.
 - `corpus_embeddings` - A 2 dimensional tensor with the corpus embeddings.
 - `query_chunk_size` - Process 100 queries simultaneously. Increasing that value increases the speed, but requires more memory.
 - `corpus_chunk_size` - Scans the corpus 100k entries at a time. Increasing that value increases the speed, but requires more memory.
 - `top_k` - Retrieve top k matching entries.
 - `score_function` - Function for computing scores. By default: cosine similarity.



Stage 5: Generation LLM





Stage 5: Generation LLM

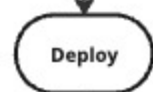
Stage 6: Conversation Summary



Stage 7: Evaluation



Stage 8 : Deployment



Stage 10: Front-End

