Comparative Study of Different Embedding Models

There are many different embedding models available, and you need to select one based on your use case. You can download a model, run the embedding process, and save the documents in encoded form. When creating a knowledge base, it's fine to run a local model on a GPU or CPU (for smaller models), but the challenge arises during retrieval. The models are large, so when performing similarity searches in the actual code, the embedding model needs to process the user query, which can be computationally expensive. Running this on a CPU can require significant power and increase costs, making APIs a comparable alternative.

Most providers offer HTTPS requests to retrieve relevant documents with embeddings. However, to reduce overall system latency, local execution is preferred. With HTTPS calls, latency depends on internet speed and the cloud server's location. For example, latency can vary significantly based on these factors.

Local Embedding Models:

- Running models locally allows for control over latency and data privacy.
- Small models can run on a CPU, but larger ones require a GPU.
- Embedding queries locally can consume significant computational resources.

Using APIs:

- APIs can offload computation to external servers, potentially reducing local resource usage.
- Latency can depend on internet speed and server location.
- API usage costs may accumulate based on request volume.

Latency Considerations:

- Local execution reduces dependency on network conditions.
- Cloud-based solutions may introduce delays, influenced by geographical distance to data centers.

Performance Optimization:

- Consider hybrid approaches, such as using local models for frequent queries and APIs for more complex tasks.
- Evaluate the cost versus performance trade-offs based on your specific use case.

Local Embedding Models

Using the local embedding model is something which can easily implemented using the Hugging face and using the langchain_huggingface embedding module here we need to provide the model name and also keep in mind to the dimensions of the embedding as the most of the open source embedding models does not provide the flexibility of changing the dimensions .

Here are some of the embedding models used in the jupyter notebook are

- **facebook/fasttext-be-vectors**

Pre-trained word vectors for Belarusian, trained on Wikipedia and Common Crawl using fastText

300 dimensional embedding is a huge model

This is something that we need to test the accuracy that it achieves as the training dataset has to be wide enough so that it can be used in many use cases . The accuracy can only be judge when the actual use case data points are used .

- **sentence-transformers/all-mpnet-base-v2**

Multilingual sentence embedding model trained on 50+ languages

Produces 768 dimensional embeddings for input sentence

This model is the default model that the hugging face has kept . these models can be fine tuned on the particular datasets but the problem is to get the dataset the most use case of the GenAI

Is around the stuff where the dataset is not present which makes it really difficult to get the good performance and accuracy as compared to the Open AI models .

These models can beat the performance of big embedding models in the particular dataset on which the dataset is good enough .

Also running these models is the biggest hurdle in the amount of compute it takes to run these on the dataset to solve a particular use case . hence the API based models services are used cause using these be more than the amount of the actual running it on a dedicated service.

Both have to be the same embedding models and the dimensions of the embedding model is what determines the accuracy as well as the latency of the overall rag . There is a default that openai uses for the embedding the Ada model that has one embedding dimension that is 1536 .

So as we know in rag the embedding model is used at two places

- Making the Embedding for the Knowledge base
- Converting the user query so that it can be used for retrieving the similar doc

So after the latest released model that provides the flexibility of the dimension of the vector . so by increasing the embedding dimension we can get more context of sentences or more schematic of the sentences .

Embedding model by OpenAI

- text -embedding-3-small
- Text-embedding-3-large
- Text-embedding-ada-002

Above two model provides the different embedding dimension that can be tuned and just varied according to time latency as well as the context of the problem statements .By default, the length of the embedding vector will be 1536 for text-embedding-3-small or 3072 for text-embedding-3-large.

Here is an analysis of time it takes to to embedded a sample statement

Elapsed time to embed text on google collab using text-embedding-3-small : 221.9877 milliseconds

Elapsed time to embed text on google collab using text-embedding-3-large : 247.7541 milliseconds

Elapsed time to embed text on google collab using text-embedding-ada-002 : 257.5264 milliseconds

As the ada-002 model can not change the embedding dimensions hence we kept it as the standard and try to see the latency of each model to embed it in the same sentence as is least in case of text-embedding-3-small . the latency does not come into the picture if the use case of chat based application. But play a vital role in voice conversation cause the latency of the bot has to be minimum else it will feel really weird to talk .

There is one more model on the LeaderBoard that is `voyage-large-2-instruct`

This is a embedding model present on the leaderboard and the usage and code implementation is very similar to the opeanai and the and the embedding can be used to implement in the similarity search . the time latency is higher as compared to text-embedding-3-small.

Note : Experiments and the code implementation is present in the in Code : embedding_models folder

- HuggingFaceembeddingmodel.ipynb
- Using_LocalModel.ipynb