Engineering Embeddings – Chapter 1 (The Basics)

# Version Information

Satya Komatineni, 2/10/24 (Started 2/9/24)

Version 2

Contents

[Version Information 1](#_Toc158459700)

[Pre-requisites 2](#_Toc158459701)

[Goal: Understand the basic aspects of engineering ai embeddings 2](#_Toc158459702)

[What this document does not address 2](#_Toc158459703)

[What are Embeddings? 2](#_Toc158459704)

[If you are in a hurry – sample code for a simple LangChain custom embedder using a cloud URL. 5](#_Toc158459705)

[If you are in a hurry – The essentials. 6](#_Toc158459706)

[Quick Example at LangChain 6](#_Toc158459707)

[Base Interface 6](#_Toc158459708)

[Examples 7](#_Toc158459709)

[High-level Observations 7](#_Toc158459710)

[More General understanding of embeddings at HF and OpenAI 7](#_Toc158459711)

[URLs of Sample implementations of LangChain embeddings 8](#_Toc158459712)

[LangChain Hugging Face custom embedder 8](#_Toc158459713)

[LangChain OpenAI custom embedder 8](#_Toc158459714)

[Fast Embeddings 8](#_Toc158459715)

[Fake Embedding 8](#_Toc158459716)

[ChromaDB Embeddings 8](#_Toc158459717)

[Understand 8](#_Toc158459718)

[Base class 9](#_Toc158459719)

[Sample Implementations 9](#_Toc158459720)

[Other Places: Sbert.net 9](#_Toc158459721)

# Pre-requisites

That you

1. are getting ready to get your feet wet with Python and RAG architectures.
2. understand Very basic LLMs.
3. heard of RAG architecture.
4. Have some idea of what Hugging Face, Lang Chain, Chromadb are.
5. Understand what embeddings are.

# Goal: Understand the basic aspects of engineering ai embeddings

This document will have:

1. A literature of where to start to understand how to obtain embeddings for your data, documents, text etc.
2. How this idea is implemented variously in Chromadb (a vector db.), Hugging Face, and Lang Chain, for they are all wrappers on the same basic idea
3. The base classes.
4. Few sample implementations in each
5. A quick sample code in python

# What this document does not address

The following important and essential topics are not discussed in this document.

1. RAG architecture in depth, where these embeddings are an integral part.
2. How the AI models that do embeddings differ in their capabilities.
3. Strategies to divide a large text document into smaller documents for accurate retrieval.
4. Sibling capabilities to embeddings to assist RAG.
5. Sentence Transformers: A key abstraction that is useful to know

# What are Embeddings?

1. Embeddings are single vector representation of a single text block.
2. Each text block is represented as a single vector (a list of numbers) over “N” number of dimensions. These dimensions usually range in the hundreds. This is done by a pretrained AI model over a large volume of data.
3. Once many such text blocks are stored as vectors in a vector database, the database can be asked to locate the nearest vectors for a given input text (once it is converted to a vector using the same scheme or model)
4. There are two main components:
   1. The pretrained AI model that can generate vectors.
   2. The vector database that can store and search for matching vectors.

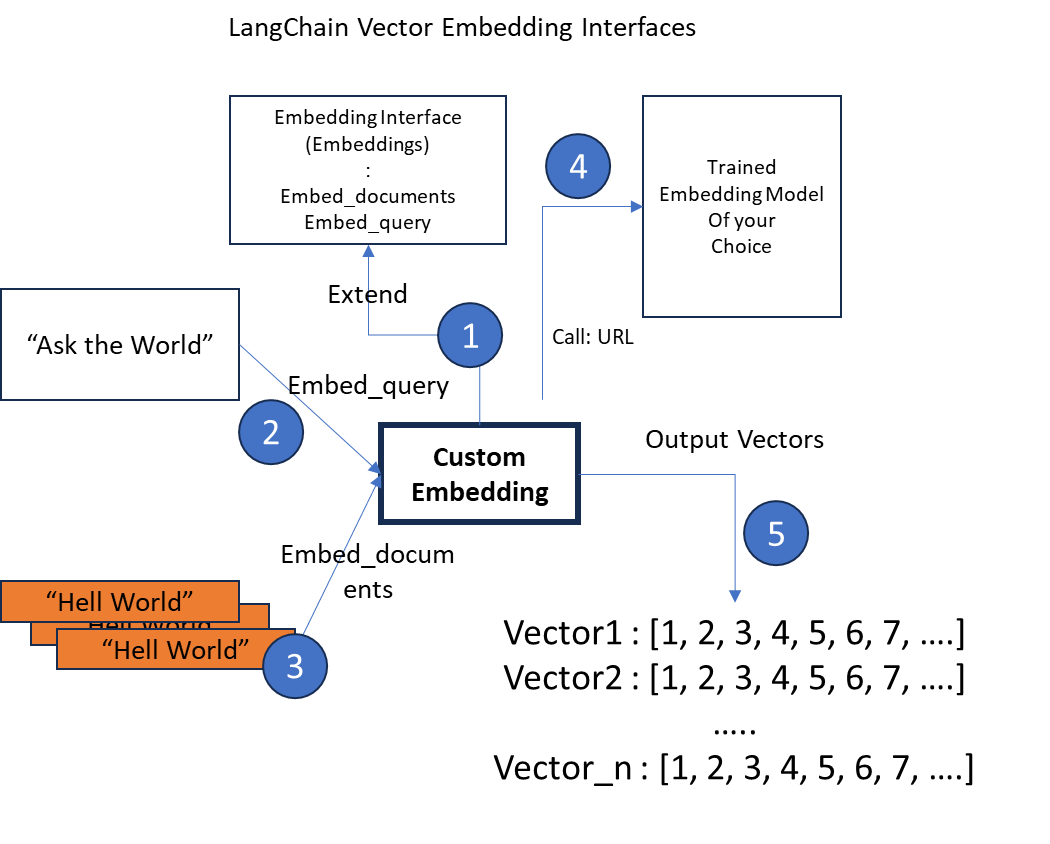
The following is an example of how the AI model is wrapped in a python library by LangChain to be used in ai python programs.

This document does not cover the RAG architecture. This need of a) vectorizing text and b) searching for that vectorized text is essential for RAG.

The following diagram shows how one can wrap an AI model for such vectorization of any text.

The following diagram shows the following steps:

1. LangChain offers a base class so that all Embeddings from all AI models can be abstracted into a similar interface irrespective of the model. This model has 2 methods.
   1. Embed\_documents: This method converts a “list” of text blocks into a list of vectors.
   2. Embed\_query: This method converts a single text block into a single vector (a list of numbers
   3. A custom class implements these methods against a desired AI model.
2. A client program that needs to look for a vector for a given text calls the embed\_query to retrieve the vector representing the text. It then uses it to ask the vector database to look for similar documents.
3. When it is time to store a large set of documents in a vector database, a client will send a list of text blocks to be vectorized.
4. The custom class calls the pretrained embedding model to satisfy both incoming calls.
5. The model returns a collection of vectors which the custom class returns to the caller.
6. The caller then stores the vectors in a specialized vector database.



# If you are in a hurry – sample code for a simple LangChain custom embedder using a cloud URL.

1. from typing import Any, Dict, List, Optional

2.

3. import requests

4. from langchain\_core.embeddings import Embeddings

5.

6. from baselib import fileutils as fileutils

7. from baselib import baselog as log

8. from baselib import aiutils as aiutils

9.

10. """

11. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12. \* class: GenURLEmbedder(Embeddings)

13. \*

14. \* A base class for implementing URL based embedders

15. \*

16. \* 1. Import from the base interface

17. \* 2. Implement the 2 necessary methods

18. \* 3. Parameterize for URL to call, API token etc.

19. \* 4. Provide a simple single threaded implementation

20. \*

21. \* Derived classes neeed to supply the init params

22. \*

23. \* See the derived class FB\_HF\_Embeddings(GenURLEmbedder)

24. \*

25. \* Note:

26. \* Uses a few utility libraries for logging, reading files,

27. \* getting URLs, Tokens etc.

28. \*

29. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

30. """

31. class GenURLEmbedder(Embeddings):

32.     url: str

33.     params: dict

34.     token: str

35.     def \_\_init\_\_(self, url: str, token: str, params: dict) :

36.         self.url = url

37.         self.params = params

38.         self.token = token

39.

40.     def embed\_documents(self, texts: List[str]) -> List[List[float]]:

41.         response = requests.post(

42.                 self.url,

43.                 json={"inputs": texts},

44.                 headers={"Authorization": f"Bearer {self.token}"}

45.             )

46.         response.raise\_for\_status()

47.         return response.json()

48.

49.     def embed\_query(self, text: str) -> List[float]:

50.         return self.embed\_documents([text])[0]

51.

52. """

53. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

54. \* class: FB\_HF\_Embeddings(GenURLEmbedder):

55. \*

56. \* An instantiable base class that uses HF model for embeddings

57. \*

58. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

59. """

60. class FB\_HF\_Embeddings(GenURLEmbedder):

61.     def \_\_init\_\_(self) :

62.         url = aiutils.getSampleEmbeddingAPI()

63.         token = aiutils.get\_FB\_HFAPIKey()

64.         params = {}

65.         super().\_\_init\_\_(url=url,token=token,params=params)

66.

67. """

68. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

69. \* Test locally

70. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

71. """

72. def testEmbedding():

73.     e = FB\_HF\_Embeddings()

74.     r = e.embed\_query("Hello World")

75.     log.ph("Sample embedding", r)

76.

77. def localTest():

78.     log.ph1("Starting local test")

79.     testEmbedding()

80.     log.ph1("End local test")

81.

82. if \_\_name\_\_ == '\_\_main\_\_':

83.     localTest()

84.

85.

86.

# If you are in a hurry – The essentials.

This section has the theory, the base interface, and a few examples that can get you started right away. If your focus is only LangChain then this section is sufficient for your browsing. The rest is more details and how something similar is done for Chromadb and using a Hugging face cloud model.

## Quick Example at LangChain

Understand first, text embedding as a general topic in LangChain documents:

<https://python.langchain.com/docs/modules/data_connection/text_embedding/>

This has the following:

1. Use a custom OpenAI embedding.
2. Embed a collection of docs.
3. Embed a single doc.

## Base Interface

Then know the base interface of an embedding class.

You will find the LangChain custom embedding class interface here:

<https://github.com/langchain-ai/langchain/blob/a210a8bc53f9c5c4820f45abdb2cc3200d52b1e2/libs/core/langchain_core/embeddings.py#L7>

## Examples

Various LangChain custom embedding implementations including OpenAI and a few useful ones for testing are here:

<https://github.com/langchain-ai/langchain/tree/00a09e1b7117f3bde14a44748510fcccc95f9de5/libs/community/langchain_community/embeddings>

# High-level Observations

Each eco system like ChromaDB, Hugging Face, and LangChain have their own abstractions to the same basic concept.

As you implement the base interface know that the key implementation examples have the following aspects:

1. Many seem to spin off multiple threads.
2. Many seem to use async calls.
3. Straight forward single thread implementations might be slow.
4. Some use local models
5. Some use cloud APIs

The general interface has the following contracts:

1. Given a document chunk, or a single question return a list of floats.
2. Given a list of document chunks return a List of List of floats.

With that lets turn to the details.

# More General understanding of embeddings at HF and OpenAI

Get started with hugging face take on embeddings using an API:

<https://huggingface.co/blog/getting-started-with-embeddings>

You will find here:

1. This is an HF blog.
2. Sentence transformers as an example of embeddings.
3. Uses an API end point.
4. Save embeddings as a dataset for reuse.

and Open AI embeddings intro at:

<https://platform.openai.com/docs/guides/embeddings/what-are-embeddings>

what do you find here:

1. It is from OpenAI
2. Various uses of embeddings: Search, Clustering, Anomalies, diversity, classification
3. How to use their API for getting embeddings
4. Available embedding models
5. Usecases with examples
6. This is an excellent source for anyone thinking about embeddings.

# URLs of Sample implementations of LangChain embeddings

## LangChain Hugging Face custom embedder:

<https://github.com/langchain-ai/langchain/blob/00a09e1b7117f3bde14a44748510fcccc95f9de5/libs/community/langchain_community/embeddings/huggingface.py>

## LangChain OpenAI custom embedder:

<https://github.com/langchain-ai/langchain/blob/00a09e1b7117f3bde14a44748510fcccc95f9de5/libs/community/langchain_community/embeddings/openai.py>

## Fast Embeddings

<https://github.com/langchain-ai/langchain/blob/00a09e1b7117f3bde14a44748510fcccc95f9de5/libs/community/langchain_community/embeddings/fastembed.py>

## Fake Embedding

<https://github.com/langchain-ai/langchain/blob/00a09e1b7117f3bde14a44748510fcccc95f9de5/libs/community/langchain_community/embeddings/fake.py>

# ChromaDB Embeddings

ChromaDB is a popular free open source on premise vector database. It uses SQLite behind the scenes. It can be downloaded as a pip install.

Being a vector database it has some built-in embedders.

It also has a custom interface for custom embedders.

ChromaDB has a dozen or so out of the box embedders that can be registered with ChromaDB database instance to do some of the embedding automatically.

## Understand

To understand their take on embedders, start here:

<https://docs.trychroma.com/embeddings>

You will find here:

1. Available embedders
2. Default embedder.
3. Sentence transformers.
4. How to do custom chroma embeddings

## Base class

Chroma base class is at

<https://github.com/chroma-core/chroma/blob/c665838b0d143e2c2ceb82c4ade7404dc98124ff/chromadb/api/types.py#L183>

## Sample Implementations

Sample implementations in Chromadb for various embeddings are at

<https://github.com/chroma-core/chroma/blob/c665838b0d143e2c2ceb82c4ade7404dc98124ff/chromadb/utils/embedding_functions.py>

# Other Places: Sbert.net

The main link is at:

<https://www.sbert.net/docs/pretrained_models.html>

1. Various pre-trained models
2. How they use Sentence Transformers
3. Various models with their perf metrics
4. Semantic search
5. Multi-qa models
6. Multi-lingual models
7. Image and text
8. ...and more

A deep read.