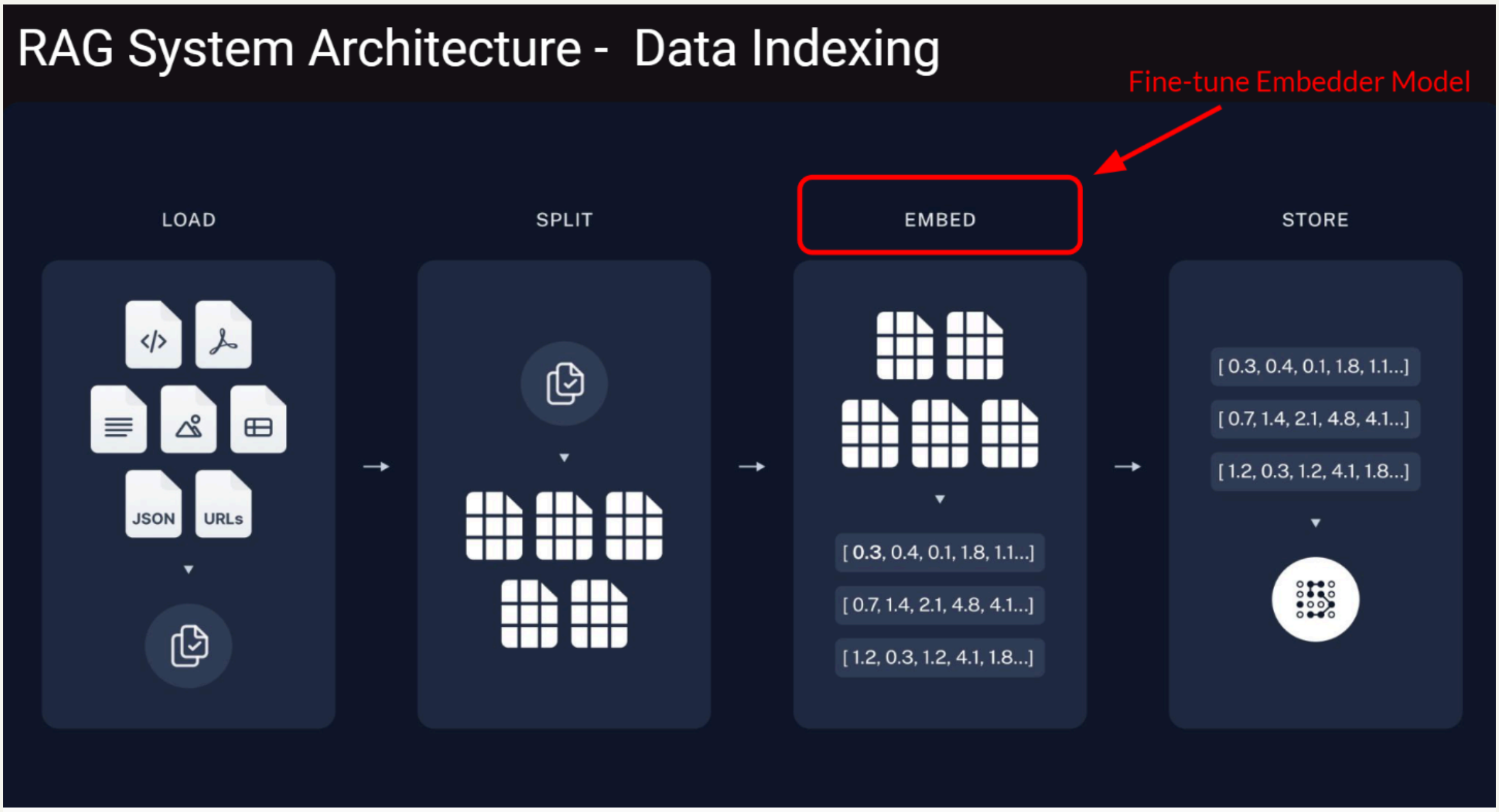
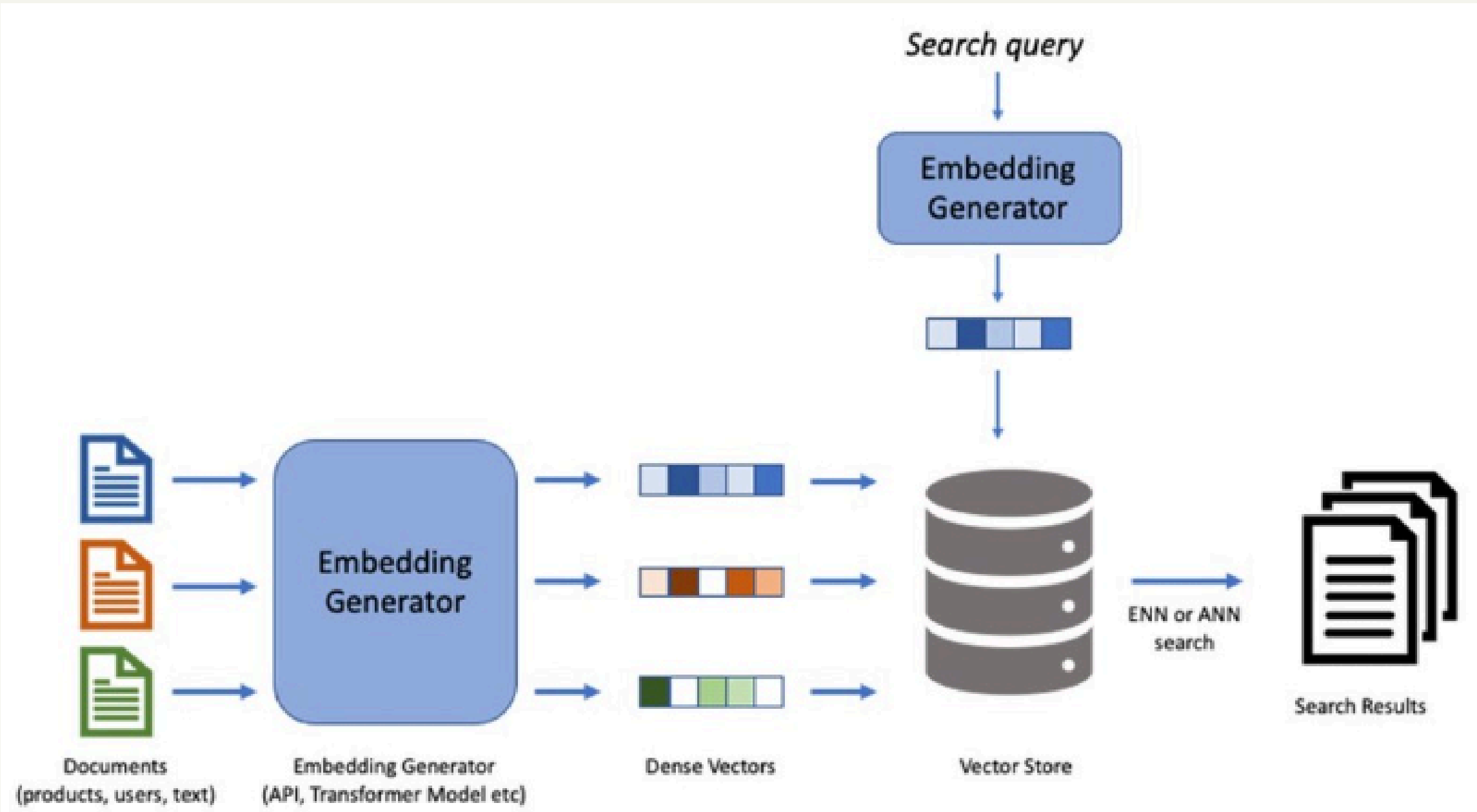
Practical Guide to Fine-tuning Embedding Models for RAG Systems



Dipanjan (DJ)

Why Fine-Tune Embedding Models?



- **Most publicly available embedding models are trained on public datasets**
- **Embedding representations may not be as contextual as expected for very custom or niche data**
- **The model might benefit from being trained on your private enterprise data and learn better representations**
- **Can align embedding model to learn good representations by training on Q-A pairs also.**
- **Can improve retrieval performance in your RAG System**

How to Fine-tune Embedding Models

We will be using the very useful Sentence Transformers library to fine-tune our embedder model. We will also use an open embedder model so you can host it anywhere.

The key components and steps in the fine-tuning process are depicted in the following figure:

- **Step 1: Prepare your dataset for training the model**
- **Step 2: Load Pre-trained Embedding Model and Choose your Loss Function**
- **Step 3: Setup Training Arguments**
- **Step 4: Build an evaluator for evaluating the model on validation data (optional)**
- **Step 5: Train the Embedding Model**

Created by: Dipanjan (DJ)

Step 1 - Dataset Preparation

Load Prepared Dataset

We already prepared our data for RAG in the previous demo, lets load up that dataset of context, question and answer triplets.

In [1]:
```
from datasets import load_dataset

dataset = load_dataset("dipanjanS/india_rag_wikidata", split="train")
```

In [2]:
```
dataset
```

Out[2]:
```
Dataset({
    features: ['article_id', 'context', 'question', 'answer'],
    num_rows: 1340
})
```

In [3]:
```
import pandas as pd
df = pd.DataFrame(dataset)
df.head(10)
```

Out[3]:

	article_id	context	question	answer
0	73985	Basil ("Ocimum basilicum") (or) is a plant o...	What is Basil commonly known as?	Dear Friend, Basil is commonly known as Sweet ...
1	73985	Basil ("Ocimum basilicum") (or) is a plant o...	Where is Basil originally native to?	Dear Friend, Basil is originally native to Ind...

In [5]:
```
data_df = df[['question', 'context']]
data_df.head()
```

Out[5]:

	question	context
0	What is Basil commonly known as?	Basil ("Ocimum basilicum") (or) is a plant o...
1	Where is Basil originally native to?	Basil ("Ocimum basilicum") (or) is a plant o...
2	What is the significance of the Roerich Pact?	The Roerich Pact is a treaty on Protection of ...
3	When was the Roerich Pact signed?	The Roerich Pact is a treaty on Protection of ...
4	What is the history of the Indian Air Force?	The Indian Air Force is the air arm of the Ind...

In [6]:
```
df_train = data_df.reset_index(drop=True)
df_eval = df_train.sample(100).reset_index(drop=True)
```

In [7]:
```
df_train.shape, df_eval.shape
```

Out[7]:
```
((1340, 2), (100, 2))
```

In [8]:
```
from datasets import Dataset

train_dataset = Dataset.from_pandas(df_train)
eval_dataset = Dataset.from_pandas(df_eval)
```

Step 2 - Load Pre-Trained Embedding Model

Load Pre-trained Embedding Model

We load up one of the top open emebdder models, which has been trained on a lot of data from the web already

In [11]:
```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer(
    "BAAI/bge-base-en-v1.5",
)
model
```

```
modules.json:   0%|          | 0.00/349 [00:00<?, ?B/s]
config_sentence_transformers.json:   0%|          | 0.00/124 [00:00<?, ?B/s]
README.md:   0%|          | 0.00/94.6k [00:00<?, ?B/s]
sentence_bert_config.json:   0%|          | 0.00/52.0 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/777 [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/438M [00:00<?, ?B/s]
tokenizer_config.json:   0%|          | 0.00/366 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/711k [00:00<?, ?B/s]
special_tokens_map.json:   0%|          | 0.00/125 [00:00<?, ?B/s]
1_Pooling/config.json:   0%|          | 0.00/190 [00:00<?, ?B/s]
```

Out[11]:
```
SentenceTransformer(
  (0): Transformer({'max_seq_length': 512, 'do_lower_case': True}) with Transformer model: BertModel
  (1): Pooling({'word_embedding_dimension': 768, 'pooling_mode_cls_token': True, 'pooling_mode_mean_tokens': False, 'pooling
_mode_max_tokens': False, 'pooling_mode_mean_sqrt_len_tokens': False, 'pooling_mode_weightedmean_tokens': False, 'pooling_mo
de_lasttoken': False, 'include_prompt': True})
  (2): Normalize()
)
```

- **We use a popular open-source model BGE-base-en-v1.5**
- **You can choose any embedding model for this**
- **All credits to Sentence Transformers for creating useful interfaces to load, train and use these models**

Step 2 - Choose Loss Function

Define Loss Function

Here we define the MultipleNegativesRankingLoss function to be used in our model

This loss expects as input a batch consisting of sentence pairs `(a_1, p_1), (a_2, p_2)..., (a_n, p_n)` where we assume that `(a_i, p_i)` are a positive pair and `(a_i, p_j)` for `i != j` a negative pair.

For each `a_i`, it uses all other `p_j` as negative samples, i.e., for `a_i`, we have 1 positive example (`p_i`) and `n-1` negative examples (`p_j`). It then minimizes the negative log-likehood for softmax normalized scores.

In [15]:
```
from sentence_transformers.losses import MultipleNegativesRankingLoss

loss = MultipleNegativesRankingLoss(model)
loss
```

Out[15]:
```
MultipleNegativesRankingLoss(
  (model): SentenceTransformer(
    (0): Transformer({'max_seq_length': 512, 'do_lower_case': True}) with Transformer model: BertModel
    (1): Pooling({'word_embedding_dimension': 768, 'pooling_mode_cls_token': True, 'pooling_mode_mean_tokens': False, 'pooli
ng_mode_max_tokens': False, 'pooling_mode_mean_sqrt_len_tokens': False, 'pooling_mode_weightedmean_tokens': False, 'pooling_
mode_lasttoken': False, 'include_prompt': True})
    (2): Normalize()
  )
  (cross_entropy_loss): CrossEntropyLoss()
)
```

- **Loss functions quantify how well a model performs for a given batch of data, allowing an optimizer to update the model weights to produce more favourable (i.e., lower) loss values. This is the core of the training process.**

- **Sadly, there is no single loss function that works best for all use-cases. Instead, which loss function to use greatly depends on your available data and on your target task.**

- **MultipleNegativesRankingLoss is a great loss function if you only have positive pairs, for example, only pairs of similar texts like pairs of paraphrases, pairs of duplicate questions, pairs of (query, response), or (query, context)**

Step 3 - Setup Training Arguments

Here we use a slightly lower learning rate to prevent from making huge gradient updates and destroying already learnt embeddings.

Idea is to slowly align the embeddings based on our current domain and data with small gradient updates.

It is usually recommended to train on as much good quality data as possible especially for an embedder model as compared to fine-tuning an LLM

In [18]:

```
from sentence_transformers import SentenceTransformerTrainingArguments
from sentence_transformers.training_args import BatchSamplers

args = SentenceTransformerTrainingArguments(
    # Required parameter:
    output_dir="bge-base-runs",
    # Optional training parameters:
    max_steps=332,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=3e-6,
    warmup_ratio=0.1,
    fp16=True,  # Set to False if you get an error that your GPU can't run on FP16
    bf16=False,  # Set to True if you have a GPU that supports BF16
    batch_sampler=BatchSamplers.NO_DUPLICATES,  # MultipleNegativesRankingLoss benefits from no duplicate samples in a batch
    # Optional tracking/debugging parameters:
    eval_strategy="steps",
    eval_steps=20,
    save_strategy="steps",
    save_steps=100,
    save_total_limit=2,
    logging_steps=20,
)
```

- **max_steps will be the total number of batches which the model will go through when training**
- **learning_rate is useful when gradient updates are happening, do not set this to a very high number as it can destroy what the model has previously learnt**
- **batch_size is the number of records which will pass through the model before a full gradient update happens through backpropagation and model weights are updated**

Step 4 - Setup Trainer & Fine-tune Embedding Model

In [19]:
```
from sentence_transformers import SentenceTransformerTrainer

trainer = SentenceTransformerTrainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    loss=loss,
)
```

```
Detected kernel version 5.4.0, which is below the recommended minimum of 5.5.0; this can cause the process to hang. It is rec
ommended to upgrade the kernel to the minimum version or higher.
max_steps is given, it will override any value given in num_train_epochs
```

Fine-tune Embedder Model

In [20]:
```
trainer.train()
```

[332/332 00:57, Epoch 3/4]

Step	Training Loss	Validation Loss
20	0.184000	0.060078
40	0.112800	0.036748
60	0.098200	0.027188
80	0.051100	0.022855
100	0.066000	0.016294
120	0.046000	0.014256
140	0.043800	0.013232

- **trainer.train() will start the training (fine-tuning) process for your pre-trained embedding model**
- **In case of GPU OOM errors consider using LoRA (Parameter efficient fine-tuning) or reduce the batch size**

Bonus: PEFT for Embedding Models

```python
from peft import LoraConfig, TaskType
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("BAAI/bge-base-en-v1.5")

# Apply a PEFT Adapter
peft_config = LoraConfig(
    task_type=TaskType.FEATURE_EXTRACTION,
    inference_mode=False,
    r=8,
    lora_alpha=32,
    lora_dropout=0.1,
)
model.add_adapter(peft_config, "dense")

# add loss, training arguments and continue as before
...
...
```

- **After saving the model on the disk or cloud you can access it anytime in the future**

- **You can load your model using any libraries including transformers, sentence transformers, LangChain and more and start using them by connecting to your vector database just like any embedding model**