[![Logo](../../../_static/logo.png)](../../../index.html)

Getting Started

Sentence Transformer

rmer-models)

* [Sentence Transformer Model from a Transformers Model](../../../docs/sentence_transformer/usage/custom_models.html#sentence-transformer-model-from-a-transformers-model)

* [Pretrained Models](../../../docs/sentence_transformer/pretrained_models.html)

  * [Original Models](../../../docs/sentence_transformer/pretrained_models.html#original-models)

  * [Semantic Search Models](../../../docs/sentence_transformer/pretrained_models.html#semantic-search-models)

    * [Multi-QA Models](../../../docs/sentence_transformer/pretrained_models.html#multi-qa-models)

    * [MSMARCO Passage Models](../../../docs/sentence_transformer/pretrained_models.html#msmarco-passage-models)

  * [Multilingual Models](../../../docs/sentence_transformer/pretrained_models.html#multilingual-models)

    * [Semantic Similarity Models](../../../docs/sentence_transformer/pretrained_models.html#semantic-similarity-models)

    * [Bitext Mining](../../../docs/sentence_transformer/pretrained_models.html#bitext-mining)

  * [Image & Text-Models](../../../docs/sentence_transformer/pretrained_models.html#image-text-models)

  * [INSTRUCTOR models](../../../docs/sentence_transformer/pretrained_models.html#instructor-models)

  * [Scientific Similarity Models](../../../docs/sentence_transformer/pretrained_models.html#scientific-similarity-models)

* [Training Overview](../../../docs/sentence_transformer/training_overview.html)

  * [Why Finetune?](../../../docs/sentence_transformer/training_overview.html#why-finetune)

  * [Training Components](../../../docs/sentence_transformer/training_overview.html#training-components)

  * [Dataset](../../../docs/sentence_transformer/training_overview.html#dataset)

* [MS MARCO](../../../docs/cross_encoder/pretrained_models.html#ms-marco)
* [SQuAD (QNLI)](../../../docs/cross_encoder/pretrained_models.html#squad-qnli)
* [STSbenchmark](../../../docs/cross_encoder/pretrained_models.html#stsbenchmark)
* [Quora Duplicate Questions](../../../docs/cross_encoder/pretrained_models.html#quora-duplicate-questions)
* [NLI](../../../docs/cross_encoder/pretrained_models.html#nli)
* [Community Models](../../../docs/cross_encoder/pretrained_models.html#community-models)

* [Training Overview](../../../docs/cross_encoder/training_overview.html)
* [Training Examples](../../../docs/cross_encoder/training/examples.html)
  * [MS MARCO](../../training/ms_marco/cross_encoder_README.html)
    * [Cross-Encoder](../../training/ms_marco/cross_encoder_README.html#cross-encoder)
    * [Cross-Encoder Knowledge Distillation](../../training/ms_marco/cross_encoder_README.html#cross-encoder-knowledge-distillation)

Package Reference

* [Sentence Transformer](../../../docs/package_reference/sentence_transformer/index.html)
  * [SentenceTransformer](../../../docs/package_reference/sentence_transformer/SentenceTransformer.html)
  * [SentenceTransformer](../../../docs/package_reference/sentence_transformer/SentenceTransformer.html#id1)
  * [SentenceTransformerModelCardData](../../../docs/package_reference/sentence_transformer/SentenceTransformer.html#sentencetransformermodelcarddata)

*

[SimilarityFunction](../../../docs/package_reference/sentence_transformer/SentenceTransformer.html#similarityfunction)

* [Trainer](../../../docs/package_reference/sentence_transformer/trainer.html)

*

[SentenceTransformerTrainer](../../../docs/package_reference/sentence_transformer/trainer.html#sentencetransformertrainer)

* [Training Arguments](../../../docs/package_reference/sentence_transformer/training_args.html)

*

[SentenceTransformerTrainingArguments](../../../docs/package_reference/sentence_transformer/training_args.html#sentencetransformertrainingarguments)

* [Losses](../../../docs/package_reference/sentence_transformer/losses.html)

*

[BatchAllTripletLoss](../../../docs/package_reference/sentence_transformer/losses.html#batchalltripletloss)

*

[BatchHardSoftMarginTripletLoss](../../../docs/package_reference/sentence_transformer/losses.html#batchhardsoftmargintripletloss)

*

[BatchHardTripletLoss](../../../docs/package_reference/sentence_transformer/losses.html#batchhardtripletloss)

*

[BatchSemiHardTripletLoss](../../../docs/package_reference/sentence_transformer/losses.html#batchsemihardtripletloss)

*

[ContrastiveLoss](../../../docs/package_reference/sentence_transformer/losses.html#contrastiveloss)

*

[OnlineContrastiveLoss](../../../docs/package_reference/sentence_transformer/losses.html#onlineco
ntrastiveloss)

*

[ContrastiveTensionLoss](../../../docs/package_reference/sentence_transformer/losses.html#contras
tivetensionloss)

*

[ContrastiveTensionLossInBatchNegatives](../../../docs/package_reference/sentence_transformer/lo
sses.html#contrastivetensionlossinbatchnegatives)

    * [CoSENTLoss](../../../docs/package_reference/sentence_transformer/losses.html#cosentloss)
    * [AnglELoss](../../../docs/package_reference/sentence_transformer/losses.html#angleloss)

*

[CosineSimilarityLoss](../../../docs/package_reference/sentence_transformer/losses.html#cosinesimil
arityloss)

*

[DenoisingAutoEncoderLoss](../../../docs/package_reference/sentence_transformer/losses.html#den
oisingautoencoderloss)

*

[GISTEmbedLoss](../../../docs/package_reference/sentence_transformer/losses.html#gistembedloss
)

*

[CachedGISTEmbedLoss](../../../docs/package_reference/sentence_transformer/losses.html#cache
dgistembedloss)

    * [MSELoss](../../../docs/package_reference/sentence_transformer/losses.html#mseloss)

*

[MarginMSELoss](../../../docs/package_reference/sentence_transformer/losses.html#marginmseloss
)

*

[MatryoshkaLoss](../../../docs/package_reference/sentence_transformer/losses.html#matryoshkaloss
)

*

[Matryoshka2dLoss](../../../docs/package_reference/sentence_transformer/losses.html#matryoshka2
dloss)

*

[AdaptiveLayerLoss](../../../docs/package_reference/sentence_transformer/losses.html#adaptivelaye
rloss)

*

[MegaBatchMarginLoss](../../../docs/package_reference/sentence_transformer/losses.html#megabat
chmarginloss)

*

[MultipleNegativesRankingLoss](../../../docs/package_reference/sentence_transformer/losses.html#
multiplenegativesrankingloss)

*

[CachedMultipleNegativesRankingLoss](../../../docs/package_reference/sentence_transformer/losse
s.html#cachedmultiplenegativesrankingloss)

*

[MultipleNegativesSymmetricRankingLoss](../../../docs/package_reference/sentence_transformer/los
ses.html#multiplenegativessymmetricrankingloss)

*

[CachedMultipleNegativesSymmetricRankingLoss](../../../docs/package_reference/sentence_transfo
rmer/losses.html#cachedmultiplenegativessymmetricrankingloss)

   * [SoftmaxLoss](../../../docs/package_reference/sentence_transformer/losses.html#softmaxloss)
   * [TripletLoss](../../../docs/package_reference/sentence_transformer/losses.html#tripletloss)
  * [Samplers](../../../docs/package_reference/sentence_transformer/sampler.html)

*

[BatchSamplers](../../../docs/package_reference/sentence_transformer/sampler.html#batchsamplers
)

*

[MultiDatasetBatchSamplers](../../../docs/package_reference/sentence_transformer/sampler.html#m
ultidatasetbatchsamplers)

* [Evaluation](../../../docs/package_reference/sentence_transformer/evaluation.html)

*

[BinaryClassificationEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html
#binaryclassificationevaluator)

*

[EmbeddingSimilarityEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.ht
ml#embeddingsimilarityevaluator)

*

[InformationRetrievalEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.htm
l#informationretrievalevaluator)

*

[NanoBEIREvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#nanobe
irevaluator)

*

[MSEEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#mseevaluator
)

*

[ParaphraseMiningEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#
paraphraseminingevaluator)

*

[RerankingEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#rerankin
gevaluator)

*

[SentenceEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#sentenceevaluator)

*

[SequentialEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#sequentialevaluator)

*

[TranslationEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#translationevaluator)

*

[TripletEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#tripletevaluator)

* [Datasets](../../../docs/package_reference/sentence_transformer/datasets.html)

*

[ParallelSentencesDataset](../../../docs/package_reference/sentence_transformer/datasets.html#parallelsentencesdataset)

*

[SentenceLabelDataset](../../../docs/package_reference/sentence_transformer/datasets.html#sentencelabeldataset)

*

[DenoisingAutoEncoderDataset](../../../docs/package_reference/sentence_transformer/datasets.html#denoisingautoencoderdataset)

*

[NoDuplicatesDataLoader](../../../docs/package_reference/sentence_transformer/datasets.html#noduplicatesdataloader)

* [Models](../../../docs/package_reference/sentence_transformer/models.html)

* [Main

nevaluator)

      * [CEF1Evaluator](../../../docs/package_reference/cross_encoder/evaluation.html#cef1evaluator)

*

[CESoftmaxAccuracyEvaluator](../../../docs/package_reference/cross_encoder/evaluation.html#ceso

ftmaxaccuracyevaluator)

*

[CERerankingEvaluator](../../../docs/package_reference/cross_encoder/evaluation.html#cereranking

evaluator)

  * [util](../../../docs/package_reference/util.html)

    * [Helper Functions](../../../docs/package_reference/util.html#module-sentence_transformers.util)

*

[`community_detection()`](../../../docs/package_reference/util.html#sentence_transformers.util.comm

unity_detection)

      * [`http_get()`](../../../docs/package_reference/util.html#sentence_transformers.util.http_get)

*

[`is_training_available()`](../../../docs/package_reference/util.html#sentence_transformers.util.is_train

ing_available)

*

[`mine_hard_negatives()`](../../../docs/package_reference/util.html#sentence_transformers.util.mine_

hard_negatives)

*

[`normalize_embeddings()`](../../../docs/package_reference/util.html#sentence_transformers.util.nor

malize_embeddings)

*

[`paraphrase_mining()`](../../../docs/package_reference/util.html#sentence_transformers.util.paraphr

ase_mining)

*

[`semantic_search()`](../../../docs/package_reference/util.html#sentence_transformers.util.semantic_search)

*

[`truncate_embeddings()`](../../../docs/package_reference/util.html#sentence_transformers.util.truncate_embeddings)

* [Model Optimization](../../../docs/package_reference/util.html#module-sentence_transformers.backend)

*

[`export_dynamic_quantized_onnx_model()`](../../../docs/package_reference/util.html#sentence_transformers.backend.export_dynamic_quantized_onnx_model)

*

[`export_optimized_onnx_model()`](../../../docs/package_reference/util.html#sentence_transformers.backend.export_optimized_onnx_model)

*

[`export_static_quantized_openvino_model()`](../../../docs/package_reference/util.html#sentence_transformers.backend.export_static_quantized_openvino_model)

  * [Similarity Metrics](../../../docs/package_reference/util.html#module-sentence_transformers.util)

    * [`cos_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.cos_sim)

    * [`dot_score()`](../../../docs/package_reference/util.html#sentence_transformers.util.dot_score)

*

[`euclidean_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.euclidean_sim)

*

[`manhattan_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.manhattan_sim)

*

[`pairwise_cos_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.pairwise_ cos_sim)

*

[`pairwise_dot_score()`](../../../docs/package_reference/util.html#sentence_transformers.util.pairwise_dot_score)

*

[`pairwise_euclidean_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.pairwise_euclidean_sim)

*

[`pairwise_manhattan_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.pairwise_manhattan_sim)

__[Sentence Transformers](../../../index.html)

  * [](../../../index.html)
  * [Usage](../../../docs/sentence_transformer/usage/usage.html)
  * Semantic Search

* [Edit on GitHub](https://github.com/UKPLab/sentence-transformers/blob/master/examples/applications/semantic-search/README.md)

* * *

# Semantic Searchïƒ•

Semantic search seeks to improve search accuracy by understanding the semantic
meaning of the search query and the corpus to search over. Semantic search can
also perform well given synonyms, abbreviations, and misspellings, unlike

keyword search engines that can only find documents based on lexical matches.

## Backgroundïƒ•

The idea behind semantic search is to embed all entries in your corpus,

whether they be sentences, paragraphs, or documents, into a vector space. At

search time, the query is embedded into the same vector space and the closest

embeddings from your corpus are found. These entries should have a high

semantic similarity with the query.

![SemanticSearch](https://raw.githubusercontent.com/UKPLab/sentence-

transformers/master/docs/img/SemanticSearch.png)

## Symmetric vs. Asymmetric Semantic Searchïƒ•

A **critical distinction** for your setup is _symmetric_ vs. _asymmetric

semantic search_ :

* For **symmetric semantic search** your query and the entries in your corpus are of about the same length and have the same amount of content. An example would be searching for similar questions: Your query could for example be _â€œHow to learn Python online?â€•_ and you want to find an entry like _â€œHow to learn Python on the web?â€•_. For symmetric tasks, you could potentially flip the query and the entries in your corpus.

* Related training example: [Quora Duplicate Questions](../../training/quora_duplicate_questions/README.html).

* Suitable models: [Pre-Trained Sentence Embedding Models](../../../docs/sentence_transformer/pretrained_models.html)

* For **asymmetric semantic search** , you usually have a **short query** (like a question or some keywords) and you want to find a longer paragraph answering the query. An example would be a query like _â€œWhat is Pythonâ€•_ and you want to find the paragraph _â€œPython is an interpreted, high-level and general-purpose programming language. Pythonâ€™s design philosophy â€¦â€•_. For asymmetric tasks, flipping the query and the entries in your corpus usually does not make sense.

* Related training example: [MS MARCO](../../training/ms_marco/README.html)

* Suitable models: [Pre-Trained MS MARCO Models](../../../docs/pretrained-models/msmarco-v5.html)

It is critical **that you choose the right model** for your type of task.

## Manual Implementationïƒ•

For small corpora (up to about 1 million entries), we can perform semantic
search with a manual implementation by computing the embeddings for the corpus
as well as for our query, and then calculating the [semantic textual
similarity](../../../docs/sentence_transformer/usage/semantic_textual_similarity.html)
using
[`SentenceTransformer.similarity`](../../../docs/package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer.similarity
"sentence_transformers.SentenceTransformer.similarity").

For a simple example, see [semantic_search.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/applications/semantic-search/semantic_search.py):

Output

```
Query: A man is eating pasta.
Top 5 most similar sentences in corpus:
A man is eating food. (Score: 0.7035)
A man is eating a piece of bread. (Score: 0.5272)
A man is riding a horse. (Score: 0.1889)
A man is riding a white horse on an enclosed ground. (Score: 0.1047)
A cheetah is running behind its prey. (Score: 0.0980)

Query: Someone in a gorilla costume is playing a set of drums.
Top 5 most similar sentences in corpus:
A monkey is playing drums. (Score: 0.6433)
A woman is playing violin. (Score: 0.2564)
A man is riding a horse. (Score: 0.1389)
A man is riding a white horse on an enclosed ground. (Score: 0.1191)
A cheetah is running behind its prey. (Score: 0.1080)

Query: A cheetah chases prey on across a field.
```

```
Top 5 most similar sentences in corpus:

A cheetah is running behind its prey. (Score: 0.8253)

A man is eating food. (Score: 0.1399)

A monkey is playing drums. (Score: 0.1292)

A man is riding a white horse on an enclosed ground. (Score: 0.1097)

A man is riding a horse. (Score: 0.0650)
```

```
"""
This is a simple application for sentence embeddings: semantic search


We have a corpus with various sentences. Then, for a given query sentence,

we want to find the most similar sentence in this corpus.


This script outputs for various queries the top 5 most similar sentences in the corpus.
"""


import torch


from sentence_transformers import SentenceTransformer


embedder = SentenceTransformer("all-MiniLM-L6-v2")


# Corpus with example sentences
corpus = [
    "A man is eating food.",
    "A man is eating a piece of bread.",
    "The girl is carrying a baby.",
    "A man is riding a horse.",
    "A woman is playing violin.",
    "Two men pushed carts through the woods.",
    "A man is riding a white horse on an enclosed ground.",
    "A monkey is playing drums.",
    "A cheetah is running behind its prey.",
]
# Use "convert_to_tensor=True" to keep the tensors on GPU (if available)
corpus_embeddings = embedder.encode(corpus, convert_to_tensor=True)

# Query sentences:
queries = [
    "A man is eating pasta.",
    "Someone in a gorilla costume is playing a set of drums.",
    "A cheetah chases prey on across a field.",
]

# Find the closest 5 sentences of the corpus for each query sentence based on cosine similarity
top_k = min(5, len(corpus))
for query in queries:
    query_embedding = embedder.encode(query, convert_to_tensor=True)

    # We use cosine-similarity and torch.topk to find the highest 5 scores
    similarity_scores = embedder.similarity(query_embedding, corpus_embeddings)[0]
    scores, indices = torch.topk(similarity_scores, k=top_k)
```

```
print("\nQuery:", query)
print("Top 5 most similar sentences in corpus:")

for score, idx in zip(scores, indices):
    print(corpus[idx], f"(Score: {score:.4f})")

"""
# Alternatively, we can also use util.semantic_search to perform cosine similarty + topk
hits = util.semantic_search(query_embedding, corpus_embeddings, top_k=5)
hits = hits[0]      #Get the hits for the first query
for hit in hits:
    print(corpus[hit['corpus_id']], "(Score: {:.4f})".format(hit['score']))
"""
```

## Optimized Implementationïƒ•

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

The function accepts the following parameters:

sentence_transformers.util.semantic_search(_query_embeddings: ~torch.Tensor, corpus_embeddings: ~torch.Tensor, query_chunk_size: int = 100, corpus_chunk_size: int = 500000, top_k: int = 10, score_function: ~typing.Callable[[~torch.Tensor, ~torch.Tensor], ~torch.Tensor] = <function cos_sim>_) -> list[list[dict[str, int |

float]]][[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L440-L517)ïƒ•

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** (_Tensor_) â€“ A 2 dimensional tensor with the query embeddings.

* **corpus_embeddings** (_Tensor_) â€“ A 2 dimensional tensor with the corpus embeddings.

* **query_chunk_size** (_int_ _,__optional_) â€“ Process 100 queries simultaneously. Increasing that value increases the speed, but requires more memory. Defaults to 100.

* **corpus_chunk_size** (_int_ _,__optional_) â€“ Scans the corpus 100k entries at a time. Increasing that value increases the speed, but requires more memory. Defaults to 500000.

* **top_k** (_int_ _,__optional_) â€“ Retrieve top k matching entries. Defaults to 10.

* **score_function** (_Callable_ _[__[__Tensor_ _,__Tensor_ _]__,__Tensor_ _]__,__optional_) â€“ Function for computing scores. By default, cosine similarity.

Returns:

A list with one entry for each query. Each entry is a list of dictionaries

with the keys â€˜corpus_idâ€™ and â€˜scoreâ€™, sorted by decreasing cosine

similarity scores.

Return type:

List[List[Dict[str, Union[int, float]]]]

By default, up to 100 queries are processed in parallel. Further, the corpus

is chunked into set of up to 500k entries. You can increase `query_chunk_size`

and `corpus_chunk_size`, which leads to increased speed for large corpora, but

also increases the memory requirement.

## Speed Optimizationïƒ•

To get the optimal speed for the `util.semantic_search` method, it is

advisable to have the `query_embeddings` as well as the `corpus_embeddings` on

the same GPU-device. This significantly boost the performance. Further, we can

normalize the corpus embeddings so that each corpus embeddings is of length 1.

In that case, we can use dot-product for computing scores.

corpus_embeddings = corpus_embeddings.to("cuda")

corpus_embeddings = util.normalize_embeddings(corpus_embeddings)

query_embeddings = query_embeddings.to("cuda")

query_embeddings = util.normalize_embeddings(query_embeddings)

hits = util.semantic_search(query_embeddings, corpus_embeddings, score_function=util.dot_score)

## Elasticsearchïƒ•

[Elasticsearch](https://www.elastic.co/elasticsearch/) has the possibility to
[index dense vectors](https://www.elastic.co/what-is/vector-search) and to use
them for document scoring. We can easily index embedding vectors, store other
data alongside our vectors and, most importantly, efficiently retrieve
relevant entries using [approximate nearest neighbor
search](https://www.elastic.co/blog/introducing-approximate-nearest-neighbor-search-in-elasticsearch-8-0) (HNSW, see also below) on the embeddings.

For further details, see
[semantic_search_quora_elasticsearch.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/applications/semantic-search/semantic_search_quora_elasticsearch.py).

## Approximate Nearest Neighborïƒ•

Searching a large corpus with millions of embeddings can be time-consuming if

exact nearest neighbor search is used (like it is used by

`util.semantic_search`).

In that case, Approximate Nearest Neighbor (ANN) can be helpful. Here, the

data is partitioned into smaller fractions of similar embeddings. This index

can be searched efficiently and the embeddings with the highest similarity

(the nearest neighbors) can be retrieved within milliseconds, even if you have

millions of vectors. However, the results are not necessarily exact. It is

possible that some vectors with high similarity will be missed.

For all ANN methods, there are usually one or more parameters to tune that

determine the recall-speed trade-off. If you want the highest speed, you have

a high chance of missing hits. If you want high recall, the search speed

decreases.

Three popular libraries for approximate nearest neighbor are

[Annoy](https://github.com/spotify/annoy),

[FAISS](https://github.com/facebookresearch/faiss), and

[hnswlib](https://github.com/nmslib/hnswlib/).

Examples:

*

[semantic_search_quora_hnswlib.py](https://github.com/UKPLab/sentence-transformers/tree/master

/examples/applications/semantic-search/semantic_search_quora_hnswlib.py)

*

[semantic_search_quora_annoy.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/applications/semantic-search/semantic_search_quora_annoy.py)

*

[semantic_search_quora_faiss.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/applications/semantic-search/semantic_search_quora_faiss.py)

## Retrieve & Re-Rankïƒ•

For complex semantic search scenarios, a two-stage retrieve & re-rank pipeline

is advisable:

![InformationRetrieval](https://raw.githubusercontent.com/UKPLab/sentence-transformers/master/docs/img/InformationRetrieval.png)

For further details, see [Retrieve & Re-rank](../retrieve_rerank/README.html).

## Examplesïƒ•

We list a handful of common use cases:

### Similar Questions Retrievalïƒ•

[semantic_search_quora_pytorch.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/applications/semantic-search/semantic_search_quora_pytorch.py) [[Colab

version](https://colab.research.google.com/drive/12cn5Oo0v3HfQQ8Tv6-ukgxXSmT3zl35A?usp=sharing)
] shows an example based on the [Quora duplicate questions](https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs) dataset. The user can enter a question, and the code retrieves the most similar questions from the dataset using `util.semantic_search`. As model, we use [distilbert-multilingual-nli-stsb-quora-ranking](https://huggingface.co/sentence-transformers/distilbert-multilingual-nli-stsb-quora-ranking), which was trained to identify similar questions and supports 50+ languages. Hence, the user can input the question in any of the 50+ languages. This is a **symmetric search task** , as the search queries have the same length and content as the questions in the corpus.

### Similar Publication Retrievalïƒ•

[semantic_search_publications.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/applications/semantic-search/semantic_search_publications.py) [[Colab version](https://colab.research.google.com/drive/12hfBveGHRsxhPIUMmJYrll2lFU4fOX06?usp=sharing)
] shows an example how to find similar scientific publications. As corpus, we use all publications that have been presented at the EMNLP 2016 - 2018 conferences. As search query, we input the title and abstract of more recent publications and find related publications from our corpus. We use the [SPECTER](https://huggingface.co/sentence-transformers/allenai-specter) model. This is a **symmetric search task** , as the paper in the corpus consists of title & abstract and we search for title & abstract.

### Question & Answer Retrievalïƒ•

[semantic_search_wikipedia_qa.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/applications/semantic-search/semantic_search_wikipedia_qa.py) [[Colab Version](https://colab.research.google.com/drive/11GunvCqJuebfeTlgbJWkIMT0xJH6PWF1?usp=sharing)
]: This example uses a model that was trained on the [Natural Questions dataset](https://huggingface.co/datasets/sentence-transformers/natural-questions). It consists of about 100k real Google search queries, together with an annotated passage from Wikipedia that provides the answer. It is an example of an **asymmetric search task**. As corpus, we use the smaller [Simple English Wikipedia](https://simple.wikipedia.org/wiki/Main_Page) so that it fits easily into memory.

[retrieve_rerank_simple_wikipedia.ipynb](https://github.com/UKPLab/sentence-transformers/tree/master/examples/applications/semantic-search/../retrieve_rerank/retrieve_rerank_simple_wikipedia.ipynb) [[Colab Version](https://colab.research.google.com/github/UKPLab/sentence-transformers/blob/master/examples/applications/retrieve_rerank/retrieve_rerank_simple_wikipedia.ipynb)
]: This script uses the [Retrieve & Re-rank](../retrieve_rerank/README.html) strategy and is an example for an **asymmetric search task**. We split all Wikipedia articles into paragraphs and encode them with a bi-encoder. If a new query / question is entered, it is encoded by the same bi-encoder and the paragraphs with the highest cosine-similarity are retrieved. Next, the

retrieved candidates are scored by a Cross-Encoder re-ranker and the 5 passages with the highest score from the Cross-Encoder are presented to the user. We use models that were trained on the [MS Marco Passage Reranking](https://github.com/microsoft/MSMARCO-Passage-Ranking/) dataset, a dataset with about 500k real queries from Bing search.

[
Previous](../../../docs/sentence_transformer/usage/semantic_textual_similarity.html
"Semantic Textual Similarity") [Next](../retrieve_rerank/README.html
"Retrieve & Re-Rank")

* * *