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

Getting Started

* [Installation](../../installation.html)
  * [Install with pip](../../installation.html#install-with-pip)
  * [Install with Conda](../../installation.html#install-with-conda)
  * [Install from Source](../../installation.html#install-from-source)
  * [Editable Install](../../installation.html#editable-install)
  * [Install PyTorch with CUDA support](../../installation.html#install-pytorch-with-cuda-support)
* [Quickstart](../../quickstart.html)
  * [Sentence Transformer](../../quickstart.html#sentence-transformer)
  * [Cross Encoder](../../quickstart.html#cross-encoder)
  * [Next Steps](../../quickstart.html#next-steps)

Sentence Transformer

* [Usage](../../sentence_transformer/usage/usage.html)
  * [Computing Embeddings](../../../examples/applications/computing-embeddings/README.html)
    * [Initializing a Sentence Transformer Model](../../../examples/applications/computing-embeddings/README.html#initializing-a-sentence-transformer-model)
    * [Calculating Embeddings](../../../examples/applications/computing-embeddings/README.html#calculating-embeddings)
    * [Prompt Templates](../../../examples/applications/computing-embeddings/README.html#prompt-templates)

* [Input Sequence Length](../../../examples/applications/computing-embeddings/README.html#id1)
* [Multi-Process / Multi-GPU Encoding](../../../examples/applications/computing-embeddings/README.html#multi-process-multi-gpu-encoding)
* [Semantic Textual Similarity](../../sentence_transformer/usage/semantic_textual_similarity.html)
* [Similarity Calculation](../../sentence_transformer/usage/semantic_textual_similarity.html#similarity-calculation)
* [Semantic Search](../../../examples/applications/semantic-search/README.html)
* [Background](../../../examples/applications/semantic-search/README.html#background)
* [Symmetric vs. Asymmetric Semantic Search](../../../examples/applications/semantic-search/README.html#symmetric-vs-asymmetric-semantic-search)
* [Manual Implementation](../../../examples/applications/semantic-search/README.html#manual-implementation)
* [Optimized Implementation](../../../examples/applications/semantic-search/README.html#optimized-implementation)
* [Speed Optimization](../../../examples/applications/semantic-search/README.html#speed-optimization)
* [Elasticsearch](../../../examples/applications/semantic-search/README.html#elasticsearch)
* [Approximate Nearest Neighbor](../../../examples/applications/semantic-search/README.html#approximate-nearest-neighbor)
* [Retrieve & Re-Rank](../../../examples/applications/semantic-search/README.html#retrieve-re-rank)

* [Examples](../../../examples/applications/semantic-search/README.html#examples)
* [Retrieve & Re-Rank](../../../examples/applications/retrieve_rerank/README.html)
* [Retrieve & Re-Rank Pipeline](../../../examples/applications/retrieve_rerank/README.html#retrieve-re-rank-pipeline)
* [Retrieval: Bi-Encoder](../../../examples/applications/retrieve_rerank/README.html#retrieval-bi-encoder)
* [Re-Ranker: Cross-Encoder](../../../examples/applications/retrieve_rerank/README.html#re-ranker-cross-encoder)
* [Example Scripts](../../../examples/applications/retrieve_rerank/README.html#example-scripts)
* [Pre-trained Bi-Encoders (Retrieval)](../../../examples/applications/retrieve_rerank/README.html#pre-trained-bi-encoders-retrieval)
* [Pre-trained Cross-Encoders (Re-Ranker)](../../../examples/applications/retrieve_rerank/README.html#pre-trained-cross-encoders-re-ranker)
* [Clustering](../../../examples/applications/clustering/README.html)
* [k-Means](../../../examples/applications/clustering/README.html#k-means)
* [Agglomerative Clustering](../../../examples/applications/clustering/README.html#agglomerative-clustering)
* [Fast Clustering](../../../examples/applications/clustering/README.html#fast-clustering)
* [Topic Modeling](../../../examples/applications/clustering/README.html#topic-modeling)
* [Paraphrase Mining](../../../examples/applications/paraphrase-mining/README.html)
* [`paraphrase_mining()`](../../../examples/applications/paraphrase-mining/README.html#sentence_transformers.util.paraphrase_mining)
* [Translated Sentence

Mining](../../../examples/applications/parallel-sentence-mining/README.html)

* [Margin Based Mining](../../../examples/applications/parallel-sentence-mining/README.html#margin-based-mining)

* [Examples](../../../examples/applications/parallel-sentence-mining/README.html#examples)

* [Image Search](../../../examples/applications/image-search/README.html)

* [Installation](../../../examples/applications/image-search/README.html#installation)

* [Usage](../../../examples/applications/image-search/README.html#usage)

* [Examples](../../../examples/applications/image-search/README.html#examples)

* [Embedding Quantization](../../../examples/applications/embedding-quantization/README.html)

* [Binary Quantization](../../../examples/applications/embedding-quantization/README.html#binary-quantization)

* [Scalar (int8) Quantization](../../../examples/applications/embedding-quantization/README.html#scalar-int8-quantization)

* [Additional extensions](../../../examples/applications/embedding-quantization/README.html#additional-extensions)

* [Demo](../../../examples/applications/embedding-quantization/README.html#demo)

* [Try it yourself](../../../examples/applications/embedding-quantization/README.html#try-it-yourself)

* [Speeding up Inference](../../sentence_transformer/usage/efficiency.html)

* [PyTorch](../../sentence_transformer/usage/efficiency.html#pytorch)

* [ONNX](../../sentence_transformer/usage/efficiency.html#onnx)

* [OpenVINO](../../sentence_transformer/usage/efficiency.html#openvino)

* [Benchmarks](../../sentence_transformer/usage/efficiency.html#benchmarks)

* [Creating Custom Models](../../sentence_transformer/usage/custom_models.html)

gloss)

* [Paraphrase Data](../../../examples/training/paraphrases/README.html)

* [Pre-Trained Models](../../../examples/training/paraphrases/README.html#pre-trained-models)

* [Quora Duplicate Questions](../../../examples/training/quora_duplicate_questions/README.html)

* [Training](../../../examples/training/quora_duplicate_questions/README.html#training)

* [MultipleNegativesRankingLoss](../../../examples/training/quora_duplicate_questions/README.html#multiplenegativesrankingloss)

* [Pretrained Models](../../../examples/training/quora_duplicate_questions/README.html#pretrained-models)

* [MS MARCO](../../../examples/training/ms_marco/README.html)

* [Bi-Encoder](../../../examples/training/ms_marco/README.html#bi-encoder)

* [Matryoshka Embeddings](../../../examples/training/matryoshka/README.html)

* [Use Cases](../../../examples/training/matryoshka/README.html#use-cases)

* [Results](../../../examples/training/matryoshka/README.html#results)

* [Training](../../../examples/training/matryoshka/README.html#training)

* [Inference](../../../examples/training/matryoshka/README.html#inference)

* [Code Examples](../../../examples/training/matryoshka/README.html#code-examples)

* [Adaptive Layers](../../../examples/training/adaptive_layer/README.html)

* [Use Cases](../../../examples/training/adaptive_layer/README.html#use-cases)

* [Results](../../../examples/training/adaptive_layer/README.html#results)

* [Training](../../../examples/training/adaptive_layer/README.html#training)

* [Inference](../../../examples/training/adaptive_layer/README.html#inference)

* [Code Examples](../../../examples/training/adaptive_layer/README.html#code-examples)

* [Multilingual Models](../../../examples/training/multilingual/README.html)

* [Extend your own models](../../../examples/training/multilingual/README.html#extend-your-own-models)

sentence-pairs)](../../../examples/training/data_augmentation/README.html#scenario-2-no-annotated-datasets-only-unlabeled-sentence-pairs)

* [Training](../../../examples/training/data_augmentation/README.html#training)

* [Citation](../../../examples/training/data_augmentation/README.html#citation)

* [Training with Prompts](../../../examples/training/prompts/README.html)

* [What are Prompts?](../../../examples/training/prompts/README.html#what-are-prompts)

* [Why would we train with Prompts?](../../../examples/training/prompts/README.html#why-would-we-train-with-prompts)

* [How do we train with Prompts?](../../../examples/training/prompts/README.html#how-do-we-train-with-prompts)

* [Training with PEFT Adapters](../../../examples/training/peft/README.html)

* [Compatibility Methods](../../../examples/training/peft/README.html#compatibility-methods)

* [Adding a New Adapter](../../../examples/training/peft/README.html#adding-a-new-adapter)

* [Loading a Pretrained Adapter](../../../examples/training/peft/README.html#loading-a-pretrained-adapter)

* [Training Script](../../../examples/training/peft/README.html#training-script)

* [Unsupervised Learning](../../../examples/unsupervised_learning/README.html)

* [TSDAE](../../../examples/unsupervised_learning/README.html#tsdae)

* [SimCSE](../../../examples/unsupervised_learning/README.html#simcse)

* [CT](../../../examples/unsupervised_learning/README.html#ct)

* [CT (In-Batch Negative Sampling)](../../../examples/unsupervised_learning/README.html#ct-in-batch-negative-sampling)

* [Masked Language Model (MLM)](../../../examples/unsupervised_learning/README.html#masked-language-model-mlm)

* [GenQ](../../../examples/unsupervised_learning/README.html#genq)

* [GPL](../../../examples/unsupervised_learning/README.html#gpl)

* [Performance

Comparison](../../../examples/unsupervised_learning/README.html#performance-comparison)

* [Domain Adaptation](../../../examples/domain_adaptation/README.html)

* [Domain Adaptation vs. Unsupervised Learning](../../../examples/domain_adaptation/README.html#domain-adaptation-vs-unsupervised-learning)

* [Adaptive Pre-Training](../../../examples/domain_adaptation/README.html#adaptive-pre-training)

* [GPL: Generative Pseudo-Labeling](../../../examples/domain_adaptation/README.html#gpl-generative-pseudo-labeling)

* [Hyperparameter Optimization](../../../examples/training/hpo/README.html)

* [HPO Components](../../../examples/training/hpo/README.html#hpo-components)

* [Putting It All Together](../../../examples/training/hpo/README.html#putting-it-all-together)

* [Example Scripts](../../../examples/training/hpo/README.html#example-scripts)

* [Distributed Training](../../sentence_transformer/training/distributed.html)

* [Comparison](../../sentence_transformer/training/distributed.html#comparison)

* [FSDP](../../sentence_transformer/training/distributed.html#fsdp)

Cross Encoder

* [Usage](../../cross_encoder/usage/usage.html)

* [Retrieve & Re-Rank](../../../examples/applications/retrieve_rerank/README.html)

* [Retrieve & Re-Rank Pipeline](../../../examples/applications/retrieve_rerank/README.html#retrieve-re-rank-pipeline)

* [Retrieval: Bi-Encoder](../../../examples/applications/retrieve_rerank/README.html#retrieval-bi-encoder)

* [Re-Ranker:

Cross-Encoder](../../../examples/applications/retrieve_rerank/README.html#re-ranker-cross-encoder)

* [Example Scripts](../../../examples/applications/retrieve_rerank/README.html#example-scripts)

* [Pre-trained Bi-Encoders (Retrieval)](../../../examples/applications/retrieve_rerank/README.html#pre-trained-bi-encoders-retrieval)

* [Pre-trained Cross-Encoders (Re-Ranker)](../../../examples/applications/retrieve_rerank/README.html#pre-trained-cross-encoders-re-ranker)

* [Pretrained Models](../../cross_encoder/pretrained_models.html)

* [MS MARCO](../../cross_encoder/pretrained_models.html#ms-marco)

* [SQuAD (QNLI)](../../cross_encoder/pretrained_models.html#squad-qnli)

* [STSbenchmark](../../cross_encoder/pretrained_models.html#stsbenchmark)

* [Quora Duplicate Questions](../../cross_encoder/pretrained_models.html#quora-duplicate-questions)

* [NLI](../../cross_encoder/pretrained_models.html#nli)

* [Community Models](../../cross_encoder/pretrained_models.html#community-models)

* [Training Overview](../../cross_encoder/training_overview.html)

* [Training Examples](../../cross_encoder/training/examples.html)

* [MS MARCO](../../../examples/training/ms_marco/cross_encoder_README.html)

* [Cross-Encoder](../../../examples/training/ms_marco/cross_encoder_README.html#cross-encoder)

* [Cross-Encoder Knowledge Distillation](../../../examples/training/ms_marco/cross_encoder_README.html#cross-encoder-knowledge-distillation)

Package Reference

    * [DenoisingAutoEncoderLoss](losses.html#denoisingautoencoderloss)
    * [GISTEmbedLoss](losses.html#gistembedloss)
    * [CachedGISTEmbedLoss](losses.html#cachedgistembedloss)
    * [MSELoss](losses.html#mseloss)
    * [MarginMSELoss](losses.html#marginmseloss)
    * [MatryoshkaLoss](losses.html#matryoshkaloss)
    * [Matryoshka2dLoss](losses.html#matryoshka2dloss)
    * [AdaptiveLayerLoss](losses.html#adaptivelayerloss)
    * [MegaBatchMarginLoss](losses.html#megabatchmarginloss)
    * [MultipleNegativesRankingLoss](losses.html#multiplenegativesrankingloss)
    * [CachedMultipleNegativesRankingLoss](losses.html#cachedmultiplenegativesrankingloss)
    * [MultipleNegativesSymmetricRankingLoss](losses.html#multiplenegativessymmetricrankingloss)
    * [CachedMultipleNegativesSymmetricRankingLoss](losses.html#cachedmultiplenegativessymmetricrankingloss)
    * [SoftmaxLoss](losses.html#softmaxloss)
    * [TripletLoss](losses.html#tripletloss)
  * [Samplers](sampler.html)
    * [BatchSamplers](sampler.html#batchsamplers)
    * [MultiDatasetBatchSamplers](sampler.html#multidatasetbatchsamplers)
  * [Evaluation](evaluation.html)
    * [BinaryClassificationEvaluator](evaluation.html#binaryclassificationevaluator)
    * [EmbeddingSimilarityEvaluator](evaluation.html#embeddingsimilarityevaluator)
    * [InformationRetrievalEvaluator](evaluation.html#informationretrievalevaluator)
    * [NanoBEIREvaluator](evaluation.html#nanobeirevaluator)
    * [MSEEvaluator](evaluation.html#mseevaluator)

      * [CEF1Evaluator](../cross_encoder/evaluation.html#cef1evaluator)
      * [CESoftmaxAccuracyEvaluator](../cross_encoder/evaluation.html#cesoftmaxaccuracyevaluator)
      * [CERerankingEvaluator](../cross_encoder/evaluation.html#cererankingevaluator)
  * [util](../util.html)
    * [Helper Functions](../util.html#module-sentence_transformers.util)
      * [`community_detection()`](../util.html#sentence_transformers.util.community_detection)
      * [`http_get()`](../util.html#sentence_transformers.util.http_get)
      * [`is_training_available()`](../util.html#sentence_transformers.util.is_training_available)
      * [`mine_hard_negatives()`](../util.html#sentence_transformers.util.mine_hard_negatives)
      * [`normalize_embeddings()`](../util.html#sentence_transformers.util.normalize_embeddings)
      * [`paraphrase_mining()`](../util.html#sentence_transformers.util.paraphrase_mining)
      * [`semantic_search()`](../util.html#sentence_transformers.util.semantic_search)
      * [`truncate_embeddings()`](../util.html#sentence_transformers.util.truncate_embeddings)
    * [Model Optimization](../util.html#module-sentence_transformers.backend)
      * [`export_dynamic_quantized_onnx_model()`](../util.html#sentence_transformers.backend.export_dynamic_quantized_onnx_model)
      * [`export_optimized_onnx_model()`](../util.html#sentence_transformers.backend.export_optimized_onnx_model)
      * [`export_static_quantized_openvino_model()`](../util.html#sentence_transformers.backend.export_static_quantized_openvino_model)
    * [Similarity Metrics](../util.html#module-sentence_transformers.util)
      * [`cos_sim()`](../util.html#sentence_transformers.util.cos_sim)
      * [`dot_score()`](../util.html#sentence_transformers.util.dot_score)

* [`euclidean_sim()`](../util.html#sentence_transformers.util.euclidean_sim)
* [`manhattan_sim()`](../util.html#sentence_transformers.util.manhattan_sim)
* [`pairwise_cos_sim()`](../util.html#sentence_transformers.util.pairwise_cos_sim)
* [`pairwise_dot_score()`](../util.html#sentence_transformers.util.pairwise_dot_score)
* [`pairwise_euclidean_sim()`](../util.html#sentence_transformers.util.pairwise_euclidean_sim)
* [`pairwise_manhattan_sim()`](../util.html#sentence_transformers.util.pairwise_manhattan_sim)

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

* [](../../../index.html)
* [Sentence Transformer](index.html)
* quantization

* [ Edit on GitHub](https://github.com/UKPLab/sentence-transformers/blob/master/docs/package_reference/sentence_transformer/quantization.md)

* * *

# quantizationïƒ•

`sentence_transformers.quantization` defines different helpful functions to perform embedding quantization.

Note

[Embedding Quantization](../../../examples/applications/embedding-quantization/README.html) differs from model quantization. The former shrinks

the size of embeddings such that semantic search/retrieval is faster and requires less memory and disk space. The latter refers to lowering the precision of the model weights to speed up inference. This page only shows documentation for the former.

sentence_transformers.quantization.quantize_embeddings(_embeddings : [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") | ndarray_, _precision : Literal['float32', 'int8', 'uint8', 'binary', 'ubinary']_, _ranges : ndarray | None = None_, _calibration_embeddings : ndarray | None = None_) -> ndarray[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\quantization.py#L371-L442)ïƒ•

Quantizes embeddings to a lower precision. This can be used to reduce the memory footprint and increase the speed of similarity search. The supported precisions are â€œfloat32â€•, â€œint8â€•, â€œuint8â€•, â€œbinaryâ€•, and â€œubinaryâ€•.

Parameters:

* **embeddings** â€“ Unquantized (e.g. float) embeddings with to quantize to a given precision

* **precision** â€“ The precision to convert to. Options are â€œfloat32â€•, â€œint8â€•, â€œuint8â€•, â€œbinaryâ€•, â€œubinaryâ€•.

* **ranges** (_Optional_ _[__np.ndarray_ _]_) â€“ Ranges for quantization of embeddings. This is only used for int8 quantization, where the ranges refers to the minimum and maximum values for each dimension. So, itâ€™s a 2D array with shape (2, embedding_dim). Default is None, which means that the ranges will be calculated from the calibration embeddings.

* **calibration_embeddings** (_Optional_ _[__np.ndarray_ _]_) â€“ Embeddings used for calibration during quantization. This is only used for int8 quantization, where the calibration embeddings can be used to compute ranges, i.e. the minimum and maximum values for each dimension. Default is None, which means that the ranges will be calculated from the query embeddings. This is not recommended.

Returns:

Quantized embeddings with the specified precision

sentence_transformers.quantization.semantic_search_faiss(_query_embeddings : np.ndarray_, _corpus_embeddings : np.ndarray | None = None_, _corpus_index : faiss.Index | None = None_, _corpus_precision : Literal['float32', 'uint8', 'ubinary'] = 'float32'_, _top_k : int = 10_, _ranges : np.ndarray | None = None_, _calibration_embeddings : np.ndarray | None = None_, _rescore : bool = True_, _rescore_multiplier : int = 2_, _exact : bool = True_, _output_index : bool = False_) -> tuple[list[list[dict[str, int | float]]], float, faiss.Index][[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\quantization.py#L18-L182)ï*f*•

Performs semantic search using the FAISS library.

Rescoring will be performed if: 1\. rescore is True 2\. The query embeddings
are not quantized 3\. The corpus is quantized, i.e. the corpus precision is
not float32 Only if these conditions are true, will we search for top_k *
rescore_multiplier samples and then rescore to only keep top_k.

Parameters:

* **query_embeddings** â€“ Embeddings of the query sentences. Ideally not quantized to allow for rescoring.

* **corpus_embeddings** â€“ Embeddings of the corpus sentences. Either corpus_embeddings or corpus_index should be used, not both. The embeddings can be quantized to â€œint8â€• or â€œbinaryâ€• for more efficient search.

* **corpus_index** â€“ FAISS index for the corpus sentences. Either corpus_embeddings or corpus_index should be used, not both.

* **corpus_precision** â€“ Precision of the corpus embeddings. The options are â€œfloat32â€•, â€œint8â€•, or â€œbinaryâ€•. Default is â€œfloat32â€•.

* **top_k** â€“ Number of top results to retrieve. Default is 10.

* **ranges** â€“ Ranges for quantization of embeddings. This is only used for int8 quantization,

where the ranges refers to the minimum and maximum values for each dimension. So, itâ€™s a 2D array with shape (2, embedding_dim). Default is None, which means that the ranges will be calculated from the calibration embeddings.

* **calibration_embeddings** â€“ Embeddings used for calibration during quantization. This is only used for int8 quantization, where the calibration embeddings can be used to compute ranges, i.e. the minimum and maximum values for each dimension. Default is None, which means that the ranges will be calculated from the query embeddings. This is not recommended.

* **rescore** â€“ Whether to perform rescoring. Note that rescoring still will only be used if the query embeddings are not quantized and the corpus is quantized, i.e. the corpus precision is not â€œfloat32â€•. Default is True.

* **rescore_multiplier** â€“ Oversampling factor for rescoring. The code will now search top_k * rescore_multiplier samples and then rescore to only keep top_k. Default is 2.

* **exact** â€“ Whether to use exact search or approximate search. Default is True.

* **output_index** â€“ Whether to output the FAISS index used for the search. Default is False.

Returns:

A tuple containing a list of search results and the time taken for the search.
If output_index is True, the tuple will also contain the FAISS index used for
the search.

Raises:

**ValueError** â€“ If both corpus_embeddings and corpus_index are provided or

if neither is provided.

The list of search results is in the format: [[{â€œcorpus_idâ€•: int,

â€œscoreâ€•: float}, â€¦], â€¦] The time taken for the search is a float

value.

sentence_transformers.quantization.semantic_search_usearch(_query_embeddings : np.ndarray_, _corpus_embeddings : np.ndarray | None = None_, _corpus_index : usearch.index.Index | None = None_, _corpus_precision : Literal['float32', 'int8', 'binary'] = 'float32'_, _top_k : int = 10_, _ranges : np.ndarray | None = None_, _calibration_embeddings : np.ndarray | None = None_, _rescore : bool = True_, _rescore_multiplier : int = 2_, _exact : bool = True_, _output_index : bool = False_) -> tuple[list[list[dict[str, int | float]]], float, usearch.index.Index][[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\quantization.py#L185-L368)ïƒ•

Performs semantic search using the usearch library.

Rescoring will be performed if: 1\. rescore is True 2\. The query embeddings

are not quantized 3\. The corpus is quantized, i.e. the corpus precision is

not float32 Only if these conditions are true, will we search for top_k *

rescore_multiplier samples and then rescore to only keep top_k.

Parameters:

* **query_embeddings** â€“ Embeddings of the query sentences. Ideally not quantized to allow for rescoring.

* **corpus_embeddings** â€“ Embeddings of the corpus sentences. Either corpus_embeddings or corpus_index should be used, not both. The embeddings can be quantized to â€œint8â€• or â€œbinaryâ€• for more efficient search.

* **corpus_index** â€“ usearch index for the corpus sentences. Either corpus_embeddings or corpus_index should be used, not both.

* **corpus_precision** â€“ Precision of the corpus embeddings. The options are â€œfloat32â€•, â€œint8â€•, â€œubinaryâ€• or â€œbinaryâ€•. Default is â€œfloat32â€•.

* **top_k** â€“ Number of top results to retrieve. Default is 10.

* **ranges** â€“ Ranges for quantization of embeddings. This is only used for int8 quantization, where the ranges refers to the minimum and maximum values for each dimension. So, itâ€™s a 2D array with shape (2, embedding_dim). Default is None, which means that the ranges will be calculated from the calibration embeddings.

* **calibration_embeddings** â€“ Embeddings used for calibration during quantization. This is only

used for int8 quantization, where the calibration embeddings can be used to compute ranges, i.e. the minimum and maximum values for each dimension. Default is None, which means that the ranges will be calculated from the query embeddings. This is not recommended.

* **rescore** â€“ Whether to perform rescoring. Note that rescoring still will only be used if the query embeddings are not quantized and the corpus is quantized, i.e. the corpus precision is not â€œfloat32â€•. Default is True.

* **rescore_multiplier** â€“ Oversampling factor for rescoring. The code will now search top_k * rescore_multiplier samples and then rescore to only keep top_k. Default is 2.

* **exact** â€“ Whether to use exact search or approximate search. Default is True.

* **output_index** â€“ Whether to output the usearch index used for the search. Default is False.

Returns:

A tuple containing a list of search results and the time taken for the search.
If output_index is True, the tuple will also contain the usearch index used
for the search.

Raises:

**ValueError** â€“ If both corpus_embeddings and corpus_index are provided or if neither is provided.

The list of search results is in the format: [[{â€œcorpus_idâ€•: int, â€œscoreâ€•: float}, â€¦], â€¦] The time taken for the search is a float value.

[ Previous](models.html "Models") [Next ](../cross_encoder/index.html "Cross Encoder")

* * *