[![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](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](usage/semantic_textual_similarity.html)
* [Similarity Calculation](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](usage/efficiency.html)
    * [PyTorch](usage/efficiency.html#pytorch)
    * [ONNX](usage/efficiency.html#onnx)
    * [OpenVINO](usage/efficiency.html#openvino)
    * [Benchmarks](usage/efficiency.html#benchmarks)
  * [Creating Custom Models](usage/custom_models.html)
    * [Structure of Sentence Transformer Models](usage/custom_models.html#structure-of-sentence-transformer-models)

    * [Datasets on the Hugging Face Hub](dataset_overview.html#datasets-on-the-hugging-face-hub)
    * [Pre-existing Datasets](dataset_overview.html#pre-existing-datasets)
  * [Loss Overview](loss_overview.html)
    * [Loss modifiers](loss_overview.html#loss-modifiers)
    * [Distillation](loss_overview.html#distillation)
    * [Commonly used Loss Functions](loss_overview.html#commonly-used-loss-functions)
    * [Custom Loss Functions](loss_overview.html#custom-loss-functions)
  * [Training Examples](training/examples.html)
    * [Semantic Textual Similarity](../../examples/training/sts/README.html)
      * [Training data](../../examples/training/sts/README.html#training-data)
      * [Loss Function](../../examples/training/sts/README.html#loss-function)
    * [Natural Language Inference](../../examples/training/nli/README.html)
      * [Data](../../examples/training/nli/README.html#data)
      * [SoftmaxLoss](../../examples/training/nli/README.html#softmaxloss)
      * [MultipleNegativesRankingLoss](../../examples/training/nli/README.html#multiplenegativesrankingloss)
    * [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)

* [Knowledge Distillation](../../examples/training/distillation/README.html#knowledge-distillation)
* [Speed - Performance Trade-Off](../../examples/training/distillation/README.html#speed-performance-trade-off)
* [Dimensionality Reduction](../../examples/training/distillation/README.html#dimensionality-reduction)
* [Quantization](../../examples/training/distillation/README.html#quantization)
* [Augmented SBERT](../../examples/training/data_augmentation/README.html)
* [Motivation](../../examples/training/data_augmentation/README.html#motivation)
* [Extend to your own datasets](../../examples/training/data_augmentation/README.html#extend-to-your-own-datasets)
* [Methodology](../../examples/training/data_augmentation/README.html#methodology)
* [Scenario 1: Limited or small annotated datasets (few labeled sentence-pairs)](../../examples/training/data_augmentation/README.html#scenario-1-limited-or-small-annotated-datasets-few-labeled-sentence-pairs)
* [Scenario 2: No annotated datasets (Only unlabeled 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](training/distributed.html)
  * [Comparison](training/distributed.html#comparison)
  * [FSDP](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

  * [Sentence Transformer](../package_reference/sentence_transformer/index.html)
    * [SentenceTransformer](../package_reference/sentence_transformer/SentenceTransformer.html)
    * [SentenceTransformer](../package_reference/sentence_transformer/SentenceTransformer.html#id1)
    * [SentenceTransformerModelCardData](../package_reference/sentence_transformer/SentenceTransformer.html#sentencetransformermodelcarddata)
    * [SimilarityFunction](../package_reference/sentence_transformer/SentenceTransformer.html#similarityfunction)
    * [Trainer](../package_reference/sentence_transformer/trainer.html)
    * [SentenceTransformerTrainer](../package_reference/sentence_transformer/trainer.html#sentencetransformertrainer)

* [Training Arguments](../package_reference/sentence_transformer/training_args.html)
* [SentenceTransformerTrainingArguments](../package_reference/sentence_transformer/training_args.html#sentencetransformertrainingarguments)
* [Losses](../package_reference/sentence_transformer/losses.html)
* [BatchAllTripletLoss](../package_reference/sentence_transformer/losses.html#batchalltripletloss)
* [BatchHardSoftMarginTripletLoss](../package_reference/sentence_transformer/losses.html#batchhardsoftmargintripletloss)
* [BatchHardTripletLoss](../package_reference/sentence_transformer/losses.html#batchhardtripletloss)
* [BatchSemiHardTripletLoss](../package_reference/sentence_transformer/losses.html#batchsemihardtripletloss)
* [ContrastiveLoss](../package_reference/sentence_transformer/losses.html#contrastiveloss)
* [OnlineContrastiveLoss](../package_reference/sentence_transformer/losses.html#onlinecontrastiveloss)
* [ContrastiveTensionLoss](../package_reference/sentence_transformer/losses.html#contrastivetensionloss)
* [ContrastiveTensionLossInBatchNegatives](../package_reference/sentence_transformer/losses.html#contrastivetensionlossinbatchnegatives)
* [CoSENTLoss](../package_reference/sentence_transformer/losses.html#cosentloss)

* [AnglELoss](../package_reference/sentence_transformer/losses.html#angleloss)
* [CosineSimilarityLoss](../package_reference/sentence_transformer/losses.html#cosinesimilarityloss)
* [DenoisingAutoEncoderLoss](../package_reference/sentence_transformer/losses.html#denoisingautoencoderloss)
* [GISTEmbedLoss](../package_reference/sentence_transformer/losses.html#gistembedloss)
* [CachedGISTEmbedLoss](../package_reference/sentence_transformer/losses.html#cachedgistembedloss)
* [MSELoss](../package_reference/sentence_transformer/losses.html#mseloss)
* [MarginMSELoss](../package_reference/sentence_transformer/losses.html#marginmseloss)
* [MatryoshkaLoss](../package_reference/sentence_transformer/losses.html#matryoshkaloss)
* [Matryoshka2dLoss](../package_reference/sentence_transformer/losses.html#matryoshka2dloss)
* [AdaptiveLayerLoss](../package_reference/sentence_transformer/losses.html#adaptivelayerloss)
* [MegaBatchMarginLoss](../package_reference/sentence_transformer/losses.html#megabatchmarginloss)
* [MultipleNegativesRankingLoss](../package_reference/sentence_transformer/losses.html#multiplenegativesrankingloss)
* [CachedMultipleNegativesRankingLoss](../package_reference/sentence_transformer/losses.html#cachedmultiplenegativesrankingloss)
*

[MultipleNegativesSymmetricRankingLoss](../package_reference/sentence_transformer/losses.html#multiplenegativessymmetricrankingloss)

*

[CachedMultipleNegativesSymmetricRankingLoss](../package_reference/sentence_transformer/losses.html#cachedmultiplenegativessymmetricrankingloss)

    * [SoftmaxLoss](../package_reference/sentence_transformer/losses.html#softmaxloss)

    * [TripletLoss](../package_reference/sentence_transformer/losses.html#tripletloss)

  * [Samplers](../package_reference/sentence_transformer/sampler.html)

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

*

[MultiDatasetBatchSamplers](../package_reference/sentence_transformer/sampler.html#multidatasetbatchsamplers)

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

*

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

*

[EmbeddingSimilarityEvaluator](../package_reference/sentence_transformer/evaluation.html#embeddingsimilarityevaluator)

*

[InformationRetrievalEvaluator](../package_reference/sentence_transformer/evaluation.html#informationretrievalevaluator)

*

[NanoBEIREvaluator](../package_reference/sentence_transformer/evaluation.html#nanobeirevaluator)

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

*

[ParaphraseMiningEvaluator](../package_reference/sentence_transformer/evaluation.html#paraphraseminingevaluator)
*
[RerankingEvaluator](../package_reference/sentence_transformer/evaluation.html#rerankingevaluator)
*
[SentenceEvaluator](../package_reference/sentence_transformer/evaluation.html#sentenceevaluator)
*
[SequentialEvaluator](../package_reference/sentence_transformer/evaluation.html#sequentialevaluator)
*
[TranslationEvaluator](../package_reference/sentence_transformer/evaluation.html#translationevaluator)
    * [TripletEvaluator](../package_reference/sentence_transformer/evaluation.html#tripletevaluator)
  * [Datasets](../package_reference/sentence_transformer/datasets.html)
*
[ParallelSentencesDataset](../package_reference/sentence_transformer/datasets.html#parallelsentencesdataset)
*
[SentenceLabelDataset](../package_reference/sentence_transformer/datasets.html#sentencelabeldataset)
*
[DenoisingAutoEncoderDataset](../package_reference/sentence_transformer/datasets.html#denoisingautoencoderdataset)
*
[NoDuplicatesDataLoader](../package_reference/sentence_transformer/datasets.html#noduplicatesd

ataloader)
    * [Models](../package_reference/sentence_transformer/models.html)
      * [Main Classes](../package_reference/sentence_transformer/models.html#main-classes)
      * [Further Classes](../package_reference/sentence_transformer/models.html#further-classes)
    * [quantization](../package_reference/sentence_transformer/quantization.html)
      *
[`quantize_embeddings()`](../package_reference/sentence_transformer/quantization.html#sentence_transformers.quantization.quantize_embeddings)
      *
[`semantic_search_faiss()`](../package_reference/sentence_transformer/quantization.html#sentence_transformers.quantization.semantic_search_faiss)
      *
[`semantic_search_usearch()`](../package_reference/sentence_transformer/quantization.html#sentence_transformers.quantization.semantic_search_usearch)
  * [Cross Encoder](../package_reference/cross_encoder/index.html)
    * [CrossEncoder](../package_reference/cross_encoder/cross_encoder.html)
      * [CrossEncoder](../package_reference/cross_encoder/cross_encoder.html#id1)
      * [Training Inputs](../package_reference/cross_encoder/cross_encoder.html#training-inputs)
    * [Evaluation](../package_reference/cross_encoder/evaluation.html)
      *
[CEBinaryAccuracyEvaluator](../package_reference/cross_encoder/evaluation.html#cebinaryaccuracyevaluator)
      *
[CEBinaryClassificationEvaluator](../package_reference/cross_encoder/evaluation.html#cebinaryclassificationevaluator)
      *
[CECorrelationEvaluator](../package_reference/cross_encoder/evaluation.html#cecorrelationevaluat

or)
    * [CEF1Evaluator](../package_reference/cross_encoder/evaluation.html#cef1evaluator)
*
[CESoftmaxAccuracyEvaluator](../package_reference/cross_encoder/evaluation.html#cesoftmaxaccuracyevaluator)
*
[CERerankingEvaluator](../package_reference/cross_encoder/evaluation.html#cererankingevaluator)
  * [util](../package_reference/util.html)
    * [Helper Functions](../package_reference/util.html#module-sentence_transformers.util)
*
[`community_detection()`](../package_reference/util.html#sentence_transformers.util.community_detection)
    * [`http_get()`](../package_reference/util.html#sentence_transformers.util.http_get)
*
[`is_training_available()`](../package_reference/util.html#sentence_transformers.util.is_training_available)
*
[`mine_hard_negatives()`](../package_reference/util.html#sentence_transformers.util.mine_hard_negatives)
*
[`normalize_embeddings()`](../package_reference/util.html#sentence_transformers.util.normalize_embeddings)
*
[`paraphrase_mining()`](../package_reference/util.html#sentence_transformers.util.paraphrase_mining)
*

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

*

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

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

*

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

*

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

*

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

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

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

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

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

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

*

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

*

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

*

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

*

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

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

* [](../../index.html)
* Training Overview
* [Edit on GitHub](https://github.com/UKPLab/sentence-transformers/blob/master/docs/sentence_transformer/training_overview.md)

* * *

Training Overviewïƒ•

Why Finetune?ïƒ•

Finetuning Sentence Transformer models often heavily improves the performance of the model on your use case, because each task requires a different notion of similarity. For example, given news articles:

* â€œApple launches the new iPadâ€•

* â€œNVIDIA is gearing up for the next GPU generationâ€•

Then the following use cases, we may have different notions of similarity:

* a model for **classification** of news articles as Economy, Sports, Technology, Politics, etc., should produce **similar embeddings** for these texts.

* a model for **semantic textual similarity** should produce **dissimilar embeddings** for these texts, as they have different meanings.

* a model for **semantic search** would **not need a notion for similarity** between two documents, as it should only compare queries and documents.

Also see [**Training Examples**](training/examples.html) for numerous training scripts for common real-world applications that you can adopt.

Training Componentsïƒ•

Training Sentence Transformer models involves between 3 to 5 components:

Dataset Learn how to prepare the **data** for training. Loss Function Learn how to prepare and choose a **loss** function. Training Arguments Learn which **training arguments** are useful. Evaluator Learn how to **evaluate** during and after training. Trainer Learn how to start the **training** process.

Datasetïƒ•

The `SentenceTransformerTrainer` trains and evaluates using [`datasets.Dataset`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset

"\(in datasets vmain\)") (one dataset) or

[`datasets.DatasetDict`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.DatasetDict

"\(in datasets vmain\)") instances (multiple datasets, see also Multi-dataset

training).

Data on ðŸ¤— Hugging Face Hub

If you want to load data from the [Hugging Face

Datasets](https://huggingface.co/datasets), then you should use

[`datasets.load_dataset()`](https://huggingface.co/docs/datasets/main/en/package_reference/loading_methods#datasets.load_dataset

"\(in datasets vmain\)"):

Documentation

* [Datasets, Loading from the Hugging Face Hub](https://huggingface.co/docs/datasets/main/en/loading#hugging-face-hub)

*

[`datasets.load_dataset()`](https://huggingface.co/docs/datasets/main/en/package_reference/loading_methods#datasets.load_dataset "\(in datasets vmain\)")

* [sentence-transformers/all-nli](https://huggingface.co/datasets/sentence-transformers/all-nli)

```
from datasets import load_dataset
```

```
train_dataset = load_dataset("sentence-transformers/all-nli", "pair-class", split="train")
eval_dataset = load_dataset("sentence-transformers/all-nli", "pair-class", split="dev")

print(train_dataset)
"""
Dataset({
    features: ['premise', 'hypothesis', 'label'],
    num_rows: 942069
})
"""
```

Some datasets (including [sentence-transformers/all-nli](https://huggingface.co/datasets/sentence-transformers/all-nli)) require you to provide a â€œsubsetâ€• alongside the dataset name. `sentence-transformers/all-nli` has 4 subsets, each with different data formats: [pair](https://huggingface.co/datasets/sentence-transformers/all-nli/viewer/pair), [pair-class](https://huggingface.co/datasets/sentence-transformers/all-nli/viewer/pair-class), [pair-score](https://huggingface.co/datasets/sentence-transformers/all-nli/viewer/pair-score), [triplet](https://huggingface.co/datasets/sentence-transformers/all-nli/viewer/triplet).

Note

Many Hugging Face datasets that work out of the box with Sentence Transformers have been tagged with sentence-transformers, allowing you to easily find them

by browsing to <https://huggingface.co/datasets?other=sentence-transformers>.
We strongly recommend that you browse these datasets to find training datasets
that might be useful for your tasks.

Local Data (CSV, JSON, Parquet, Arrow, SQL)

If you have local data in common file-formats, then you can load this data
easily using
[`datasets.load_dataset()`](https://huggingface.co/docs/datasets/main/en/package_reference/loadin
g_methods#datasets.load_dataset
"\(in datasets vmain\)"):

Documentation

* [Datasets, Loading local
files](https://huggingface.co/docs/datasets/main/en/loading#local-and-remote-files)
*
[`datasets.load_dataset()`](https://huggingface.co/docs/datasets/main/en/package_reference/loadin
g_methods#datasets.load_dataset "\(in datasets vmain\)")

```
from datasets import load_dataset

dataset = load_dataset("csv", data_files="my_file.csv")
```

or:

```
from datasets import load_dataset

dataset = load_dataset("json", data_files="my_file.json")
```

Local Data that requires pre-processing

If you have local data that requires some extra pre-processing, my
recommendation is to initialize your dataset using
[`datasets.Dataset.from_dict()`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset.from_dict
"\(in datasets vmain\)") and a dictionary of lists, like so:

Documentation

*

[`datasets.Dataset.from_dict()`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset.from_dict "\(in datasets vmain\)")

```
from datasets import Dataset
```

```
    anchors = []
    positives = []
    # Open a file, do preprocessing, filtering, cleaning, etc.
    # and append to the lists

    dataset = Dataset.from_dict({
        "anchor": anchors,
        "positive": positives,
    })
```

Each key from the dictionary will become a column in the resulting dataset.

Dataset Formatï*ƒ*•

It is important that your dataset format matches your loss function (or that
you choose a loss function that matches your dataset format). Verifying
whether a dataset format works with a loss function involves two steps:

1. If your loss function requires a _Label_ according to the [Loss Overview](loss_overview.html) table, then your dataset must have a **column named â€œlabelâ€• or â€œscoreâ€•**. This column is automatically taken as the label.

2. All columns not named â€œlabelâ€• or â€œscoreâ€• are considered _Inputs_ according to the [Loss Overview](loss_overview.html) table. The number of remaining columns must match the number of valid inputs for your chosen loss. The names of these columns are **irrelevant** , only the **order matters**.

For example, given a dataset with columns `["text1", "text2", "label"]` where
the â€œlabelâ€• column has float similarity score, we can use it with
[`CoSENTLoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.CoSENTLoss
"sentence_transformers.losses.CoSENTLoss"),
[`AnglELoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.AnglELoss
"sentence_transformers.losses.AnglELoss"), and
[`CosineSimilarityLoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.CosineSimilarityLoss
"sentence_transformers.losses.CosineSimilarityLoss") because it:

1. has a â€œlabelâ€• column as is required for these loss functions.

2. has 2 non-label columns, exactly the amount required by these loss functions.

Be sure to re-order your dataset columns with
[`Dataset.select_columns`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset.select_columns
"\(in datasets vmain\)") if your columns are not ordered correctly. For
example, if your dataset has `["good_answer", "bad_answer", "question"]` as
columns, then this dataset can technically be used with a loss that requires
(anchor, positive, negative) triplets, but the `good_answer` column will be
taken as the anchor, `bad_answer` as the positive, and `question` as the
negative.

Additionally, if your dataset has extraneous columns (e.g. sample_id,
metadata, source, type), you should remove these with
[`Dataset.remove_columns`](https://huggingface.co/docs/datasets/main/en/package_reference/main
_classes#datasets.Dataset.remove_columns
"\(in datasets vmain\)") as they will be used as inputs otherwise. You can
also use
[`Dataset.select_columns`](https://huggingface.co/docs/datasets/main/en/package_reference/main_
classes#datasets.Dataset.select_columns
"\(in datasets vmain\)") to keep only the desired columns.

Loss Functionï*f*•

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. See Dataset Format to learn what datasets are valid for
which loss functions. Additionally, the [Loss Overview](loss_overview.html)
will be your best friend to learn about the options.

Most loss functions can be initialized with just the `SentenceTransformer`
that youâ€™re training, alongside some optional parameters, e.g.:

Documentation

*

[`sentence_transformers.losses.CoSENTLoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.CoSENTLoss "sentence_transformers.losses.CoSENTLoss")

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

* [Loss Overview](loss_overview.html)

```
from datasets import load_dataset
from sentence_transformers import SentenceTransformer
from sentence_transformers.losses import CoSENTLoss

# Load a model to train/finetune
model = SentenceTransformer("xlm-roberta-base")

# Initialize the CoSENTLoss
# This loss requires pairs of text and a float similarity score as a label
loss = CoSENTLoss(model)

# Load an example training dataset that works with our loss function:
train_dataset = load_dataset("sentence-transformers/all-nli", "pair-score", split="train")
"""
Dataset({
    features: ['sentence1', 'sentence2', 'label'],
    num_rows: 942069
```

})

"""

Training Argumentsïƒ•

The
[`SentenceTransformerTrainingArguments`](../package_reference/sentence_transformer/training_args.html#sentence_transformers.training_args.SentenceTransformerTrainingArguments
"sentence_transformers.training_args.SentenceTransformerTrainingArguments")
class can be used to specify parameters for influencing training performance
as well as defining the tracking/debugging parameters. Although it is
optional, it is heavily recommended to experiment with the various useful
arguments.

The following are tables with some of the most useful training arguments.

Key Training Arguments for improving training performance

[`learning_rate`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.learning_rate)
[`lr_scheduler_type`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.lr_scheduler_type)
[`warmup_ratio`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.warmup_ratio)
[`num_train_epochs`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.num_train_epochs)

[`max_steps`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.max_steps)
[`per_device_train_batch_size`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.per_device_train_batch_size)
[`per_device_eval_batch_size`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.per_device_eval_batch_size)
[`auto_find_batch_size`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.auto_find_batch_size
)
[`fp16`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.fp16)
[`bf16`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.bf16)
[`gradient_accumulation_steps`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.gradient_accumulation_steps)
[`gradient_checkpointing`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.gradient_checkpointing)
[`eval_accumulation_steps`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.eval_accumulation_steps)
[`optim`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.optim)
[`batch_sampler`](../package_reference/sentence_transformer/training_args.html#sentence_transformers.training_args.SentenceTransformerTrainingArguments)
[`multi_dataset_batch_sampler`](../package_reference/sentence_transformer/training_args.html#sentence_transformers.training_args.SentenceTransformerTrainingArguments)
[`prompts`](../package_reference/sentence_transformer/training_args.html#sentence_transformers.training_args.SentenceTransformerTrainingArguments)

Key Training Arguments for observing training performance

[`eval_strategy`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.eval_strategy)
[`eval_steps`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.eval_steps)
[`save_strategy`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.save_strategy)
[`save_steps`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.save_steps)
[`save_total_limit`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.save_total_limit)
[`load_best_model_at_end`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.load_best_model_at_end)
[`report_to`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.report_to)
[`log_level`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.log_level)
[`logging_steps`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.logging_steps)
[`push_to_hub`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.push_to_hub)
[`hub_model_id`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.hub_model_id)

[`hub_strategy`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.hub_strategy)
[`hub_private_repo`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments.hub_private_repo)

Here is an example of how
[`SentenceTransformerTrainingArguments`](../package_reference/sentence_transformer/training_args.html#sentence_transformers.training_args.SentenceTransformerTrainingArguments "sentence_transformers.training_args.SentenceTransformerTrainingArguments")
can be initialized:

```
    args = SentenceTransformerTrainingArguments(
        # Required parameter:
        output_dir="models/mpnet-base-all-nli-triplet",
        # Optional training parameters:
        num_train_epochs=1,
        per_device_train_batch_size=16,
        per_device_eval_batch_size=16,
        learning_rate=2e-5,
        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,  # losses that use "in-batch negatives"
```

benefit from no duplicates

        # Optional tracking/debugging parameters:

        eval_strategy="steps",

        eval_steps=100,

        save_strategy="steps",

        save_steps=100,

        save_total_limit=2,

        logging_steps=100,

        run_name="mpnet-base-all-nli-triplet", # Will be used in W&B if `wandb` is installed

)

Evaluatorïƒ•

You can provide the

[`SentenceTransformerTrainer`](https://sbert.net/docs/package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer)

with an `eval_dataset` to get the evaluation loss during training, but it may

be useful to get more concrete metrics during training, too. For this, you can

use evaluators to assess the modelâ€™s performance with useful metrics before,

during, or after training. You can use both an `eval_dataset` and an

evaluator, one or the other, or neither. They evaluate based on the

`eval_strategy` and `eval_steps` Training Arguments.

Here are the implemented Evaluators that come with Sentence Transformers:

Evaluator | Required Data

---|---

[`BinaryClassificationEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.BinaryClassificationEvaluator "sentence_transformers.evaluation.BinaryClassificationEvaluator") | Pairs with class labels.

[`EmbeddingSimilarityEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.EmbeddingSimilarityEvaluator "sentence_transformers.evaluation.EmbeddingSimilarityEvaluator") | Pairs with similarity scores.

[`InformationRetrievalEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.InformationRetrievalEvaluator "sentence_transformers.evaluation.InformationRetrievalEvaluator") | Queries (qid => question), Corpus (cid => document), and relevant documents (qid => set[cid]).

[`NanoBEIREvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.NanoBEIREvaluator "sentence_transformers.evaluation.NanoBEIREvaluator") | No data required.

[`MSEEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.MSEEvaluator "sentence_transformers.evaluation.MSEEvaluator") | Source sentences to embed with a teacher model and target sentences to embed with the student model. Can be the same texts.

[`ParaphraseMiningEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.ParaphraseMiningEvaluator "sentence_transformers.evaluation.ParaphraseMiningEvaluator") | Mapping of IDs to sentences & pairs with IDs of duplicate sentences.

[`RerankingEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.RerankingEvaluator "sentence_transformers.evaluation.RerankingEvaluator") | List of `{'query': '...', 'positive': [...], 'negative': [...]}` dictionaries.

[`TranslationEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.TranslationEvaluator "sentence_transformers.evaluation.TranslationEvaluator")

| Pairs of sentences in two separate languages.

[`TripletEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.TripletEvaluator "sentence_transformers.evaluation.TripletEvaluator") | (anchor, positive, negative) pairs.

Additionally,

[`SequentialEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.SequentialEvaluator "sentence_transformers.evaluation.SequentialEvaluator") should be used to
combine multiple evaluators into one Evaluator that can be passed to the
[`SentenceTransformerTrainer`](../package_reference/sentence_transformer/trainer.html#sentence_transformers.trainer.SentenceTransformerTrainer "sentence_transformers.trainer.SentenceTransformerTrainer").

Sometimes you donâ€™t have the required evaluation data to prepare one of
these evaluators on your own, but you still want to track how well the model
performs on some common benchmarks. In that case, you can use these evaluators
with data from Hugging Face.

EmbeddingSimilarityEvaluator with STSb

Documentation

* [sentence-transformers/stsb](https://huggingface.co/datasets/sentence-transformers/stsb)

*

[`sentence_transformers.evaluation.EmbeddingSimilarityEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.EmbeddingSimilarityEvaluator

"sentence_transformers.evaluation.EmbeddingSimilarityEvaluator")

*

[`sentence_transformers.SimilarityFunction`](../package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SimilarityFunction "sentence_transformers.SimilarityFunction")

```
from datasets import load_dataset
from sentence_transformers.evaluation import EmbeddingSimilarityEvaluator, SimilarityFunction

# Load the STSB dataset (https://huggingface.co/datasets/sentence-transformers/stsb)
eval_dataset = load_dataset("sentence-transformers/stsb", split="validation")

# Initialize the evaluator
dev_evaluator = EmbeddingSimilarityEvaluator(
    sentences1=eval_dataset["sentence1"],
    sentences2=eval_dataset["sentence2"],
    scores=eval_dataset["score"],
    main_similarity=SimilarityFunction.COSINE,
    name="sts-dev",
)
# You can run evaluation like so:
# dev_evaluator(model)
```

TripletEvaluator with AllNLI

Documentation

  * [sentence-transformers/all-nli](https://huggingface.co/datasets/sentence-transformers/all-nli)
  * [`sentence_transformers.evaluation.TripletEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.TripletEvaluator "sentence_transformers.evaluation.TripletEvaluator")
  * [`sentence_transformers.SimilarityFunction`](../package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SimilarityFunction "sentence_transformers.SimilarityFunction")

```
from datasets import load_dataset
from sentence_transformers.evaluation import TripletEvaluator, SimilarityFunction

# Load triplets from the AllNLI dataset (https://huggingface.co/datasets/sentence-transformers/all-nli)
max_samples = 1000
eval_dataset = load_dataset("sentence-transformers/all-nli", "triplet", split=f"dev[:{max_samples}]")

# Initialize the evaluator
dev_evaluator = TripletEvaluator(
    anchors=eval_dataset["anchor"],
```

```
    positives=eval_dataset["positive"],
    negatives=eval_dataset["negative"],
    main_distance_function=SimilarityFunction.COSINE,
    name="all-nli-dev",
)
# You can run evaluation like so:
# dev_evaluator(model)
```

NanoBEIREvaluator

Documentation

*

[`sentence_transformers.evaluation.NanoBEIREvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.NanoBEIREvaluator "sentence_transformers.evaluation.NanoBEIREvaluator")

```
from sentence_transformers.evaluation import NanoBEIREvaluator

# Initialize the evaluator. Unlike most other evaluators, this one loads the relevant datasets
# directly from Hugging Face, so there's no mandatory arguments
dev_evaluator = NanoBEIREvaluator()
# You can run evaluation like so:
# dev_evaluator(model)
```

Warning

When using [Distributed Training](training/distributed.html), the evaluator
only runs on the first device, unlike the training and evaluation datasets,
which are shared across all devices.

Trainerïƒ•

The `SentenceTransformerTrainer` is where all previous components come
together. We only have to specify the trainer with the model, training
arguments (optional), training dataset, evaluation dataset (optional), loss
function, evaluator (optional) and we can start training. Letâ€™s have a look
at a script where all of these components come together:

Documentation

1. [`SentenceTransformer`](../package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")

2. [`SentenceTransformerModelCardData`](../package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.model_card.SentenceTransformerModelCardData "sentence_transformers.model_card.SentenceTransformerModelCardData")

3. [`load_dataset()`](https://huggingface.co/docs/datasets/main/en/package_reference/loading_methods#datasets.load_dataset "\(in datasets vmain\)")

4. [`MultipleNegativesRankingLoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.MultipleNegativesRankingLoss "sentence_transformers.losses.MultipleNegativesRankingLoss")

5. [`SentenceTransformerTrainingArguments`](../package_reference/sentence_transformer/training_args.html#sentence_transformers.training_args.SentenceTransformerTrainingArguments "sentence_transformers.training_args.SentenceTransformerTrainingArguments")

6. [`TripletEvaluator`](../package_reference/sentence_transformer/evaluation.html#sentence_transformers.evaluation.TripletEvaluator "sentence_transformers.evaluation.TripletEvaluator")

7. [`SentenceTransformerTrainer`](../package_reference/sentence_transformer/trainer.html#sentence_transformers.trainer.SentenceTransformerTrainer "sentence_transformers.trainer.SentenceTransformerTrainer")

8. [`SentenceTransformer.save_pretrained`](../package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer.save_pretrained "sentence_transformers.SentenceTransformer.save_pretrained")

[`SentenceTransformer.push_to_hub`](../package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer.push_to_hub "sentence_transformers.SentenceTransformer.push_to_hub")

* [Training Examples](training/examples)

```
from datasets import load_dataset
from sentence_transformers import (
    SentenceTransformer,
    SentenceTransformerTrainer,
    SentenceTransformerTrainingArguments,
    SentenceTransformerModelCardData,
)
from sentence_transformers.losses import MultipleNegativesRankingLoss
from sentence_transformers.training_args import BatchSamplers
from sentence_transformers.evaluation import TripletEvaluator

# 1. Load a model to finetune with 2. (Optional) model card data
model = SentenceTransformer(
    "microsoft/mpnet-base",
    model_card_data=SentenceTransformerModelCardData(
        language="en",
        license="apache-2.0",
```

```
        model_name="MPNet base trained on AllNLI triplets",
    )
)

# 3. Load a dataset to finetune on
dataset = load_dataset("sentence-transformers/all-nli", "triplet")
train_dataset = dataset["train"].select(range(100_000))
eval_dataset = dataset["dev"]
test_dataset = dataset["test"]

# 4. Define a loss function
loss = MultipleNegativesRankingLoss(model)

# 5. (Optional) Specify training arguments
args = SentenceTransformerTrainingArguments(
    # Required parameter:
    output_dir="models/mpnet-base-all-nli-triplet",
    # Optional training parameters:
    num_train_epochs=1,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=2e-5,
    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=100,
    save_strategy="steps",
    save_steps=100,
    save_total_limit=2,
    logging_steps=100,
    run_name="mpnet-base-all-nli-triplet",  # Will be used in W&B if `wandb` is installed
)

# 6. (Optional) Create an evaluator & evaluate the base model
dev_evaluator = TripletEvaluator(
    anchors=eval_dataset["anchor"],
    positives=eval_dataset["positive"],
    negatives=eval_dataset["negative"],
    name="all-nli-dev",
)
dev_evaluator(model)

# 7. Create a trainer & train
trainer = SentenceTransformerTrainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    loss=loss,
    evaluator=dev_evaluator,
```

```
    )
    trainer.train()

    # (Optional) Evaluate the trained model on the test set
    test_evaluator = TripletEvaluator(
        anchors=test_dataset["anchor"],
        positives=test_dataset["positive"],
        negatives=test_dataset["negative"],
        name="all-nli-test",
    )
    test_evaluator(model)

    # 8. Save the trained model
    model.save_pretrained("models/mpnet-base-all-nli-triplet/final")

    # 9. (Optional) Push it to the Hugging Face Hub
    model.push_to_hub("mpnet-base-all-nli-triplet")
```

Callbacksïƒ•

This Sentence Transformers trainer integrates support for various [`transformers.TrainerCallback`](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback "\(in transformers vmain\)") subclasses, such as:

*

[`WandbCallback`](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.integrations.WandbCallback "\(in transformers vmain\)") to automatically log training metrics to W&B if `wandb` is installed

*

[`TensorBoardCallback`](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.integrations.TensorBoardCallback "\(in transformers vmain\)") to log training metrics to TensorBoard if `tensorboard` is accessible.

*

[`CodeCarbonCallback`](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.integrations.CodeCarbonCallback "\(in transformers vmain\)") to track the carbon emissions of your model during training if `codecarbon` is installed.

> * Note: These carbon emissions will be included in your automatically
> generated model card.

See the Transformers
[Callbacks](https://huggingface.co/docs/transformers/main/en/main_classes/callback)
documentation for more information on the integrated callbacks and how to
write your own callbacks.

Multi-Dataset Trainingï•

The top performing models are trained using many datasets at once. Normally,
this is rather tricky, as each dataset has a different format. However,
`SentenceTransformerTrainer` can train with multiple datasets without having

to convert each dataset to the same format. It can even apply different loss functions to each of the datasets. The steps to train with multiple datasets are:

* Use a dictionary of [`Dataset`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset "\(in datasets vmain\)") instances (or a [`DatasetDict`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.DatasetDict "\(in datasets vmain\)")) as the `train_dataset` and `eval_dataset`.

* (Optional) Use a dictionary of loss functions mapping dataset names to losses. Only required if you wish to use different loss function for different datasets.

Each training/evaluation batch will only contain samples from one of the datasets. The order in which batches are samples from the multiple datasets is defined by the [`MultiDatasetBatchSamplers`](../package_reference/sentence_transformer/sampler.html#sentence_transformers.training_args.MultiDatasetBatchSamplers "sentence_transformers.training_args.MultiDatasetBatchSamplers") enum, which can be passed to the [`SentenceTransformerTrainingArguments`](../package_reference/sentence_transformer/training_args.html#sentence_transformers.training_args.SentenceTransformerTrainingArguments "sentence_transformers.training_args.SentenceTransformerTrainingArguments") via `multi_dataset_batch_sampler`. Valid options are:

* `MultiDatasetBatchSamplers.ROUND_ROBIN`: Round-robin sampling from each dataset until one is exhausted. With this strategy, itâ€™s likely that not all samples from each dataset are used,

but each dataset is sampled from equally.

* `MultiDatasetBatchSamplers.PROPORTIONAL` (default): Sample from each dataset in proportion to its size. With this strategy, all samples from each dataset are used and larger datasets are sampled from more frequently.

This multi-task training has been shown to be very effective, e.g. [Huang et al.](https://arxiv.org/pdf/2405.06932) employed [`MultipleNegativesRankingLoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.MultipleNegativesRankingLoss "sentence_transformers.losses.MultipleNegativesRankingLoss"), [`CoSENTLoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.CoSENTLoss "sentence_transformers.losses.CoSENTLoss"), and a variation on [`MultipleNegativesRankingLoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.MultipleNegativesRankingLoss "sentence_transformers.losses.MultipleNegativesRankingLoss") without in-batch negatives and only hard negatives to reach state-of-the-art performance on Chinese. They even applied [`MatryoshkaLoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.MatryoshkaLoss "sentence_transformers.losses.MatryoshkaLoss") to allow the model to produce [Matryoshka Embeddings](../../examples/training/matryoshka/README.html).

Training on multiple datasets looks like this:

Documentation

*

[`datasets.load_dataset()`](https://huggingface.co/docs/datasets/main/en/package_reference/loading_methods#datasets.load_dataset "\(in datasets vmain\)")

*

[`SentenceTransformer`](../package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")

*

[`SentenceTransformerTrainer`](../package_reference/sentence_transformer/trainer.html#sentence_transformers.trainer.SentenceTransformerTrainer "sentence_transformers.trainer.SentenceTransformerTrainer")

*

[`CoSENTLoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.CoSENTLoss "sentence_transformers.losses.CoSENTLoss")

*

[`MultipleNegativesRankingLoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.MultipleNegativesRankingLoss "sentence_transformers.losses.MultipleNegativesRankingLoss")

*

[`SoftmaxLoss`](../package_reference/sentence_transformer/losses.html#sentence_transformers.losses.SoftmaxLoss "sentence_transformers.losses.SoftmaxLoss")

* [sentence-transformers/all-nli](https://huggingface.co/datasets/sentence-transformers/all-nli)

* [sentence-transformers/stsb](https://huggingface.co/datasets/sentence-transformers/stsb)

* [sentence-transformers/quora-duplicates](https://huggingface.co/datasets/sentence-transformers/quora-duplicates)

* [sentence-transformers/natural-questions](https://huggingface.co/datasets/sentence-transformers/natural-questions)

Training Examples:

* [Quora Duplicate Questions > Multi-task learning](https://github.com/UKPLab/sentence-transformers/blob/master/examples/training/quora_duplicate_questions/training_multi-task-learning.py)

* [AllNLI + STSb > Multi-task learning](https://github.com/UKPLab/sentence-transformers/blob/master/examples/training/other/training_multi-task.py)

```
from datasets import load_dataset
from sentence_transformers import SentenceTransformer, SentenceTransformerTrainer
from sentence_transformers.losses import CoSENTLoss, MultipleNegativesRankingLoss,
```

SoftmaxLoss

```
    # 1. Load a model to finetune
    model = SentenceTransformer("bert-base-uncased")

    # 2. Load several Datasets to train with
    # (anchor, positive)
    all_nli_pair_train = load_dataset("sentence-transformers/all-nli", "pair", split="train[:10000]")
    # (premise, hypothesis) + label
                all_nli_pair_class_train     =     load_dataset("sentence-transformers/all-nli",     "pair-class",
split="train[:10000]")
    # (sentence1, sentence2) + score
                all_nli_pair_score_train     =     load_dataset("sentence-transformers/all-nli",     "pair-score",
split="train[:10000]")
    # (anchor, positive, negative)
    all_nli_triplet_train = load_dataset("sentence-transformers/all-nli", "triplet", split="train[:10000]")
    # (sentence1, sentence2) + score
    stsb_pair_score_train = load_dataset("sentence-transformers/stsb", split="train[:10000]")
    # (anchor, positive)
                quora_pair_train     =     load_dataset("sentence-transformers/quora-duplicates",     "pair",
split="train[:10000]")
    # (query, answer)
                 natural_questions_train     =     load_dataset("sentence-transformers/natural-questions",
split="train[:10000]")

    # We can combine all datasets into a dictionary with dataset names to datasets
    train_dataset = {
```

```
        "all-nli-pair": all_nli_pair_train,
        "all-nli-pair-class": all_nli_pair_class_train,
        "all-nli-pair-score": all_nli_pair_score_train,
        "all-nli-triplet": all_nli_triplet_train,
        "stsb": stsb_pair_score_train,
        "quora": quora_pair_train,
        "natural-questions": natural_questions_train,
    }

    # 3. Load several Datasets to evaluate with
    # (anchor, positive, negative)
    all_nli_triplet_dev = load_dataset("sentence-transformers/all-nli", "triplet", split="dev")
    # (sentence1, sentence2, score)
    stsb_pair_score_dev = load_dataset("sentence-transformers/stsb", split="validation")
    # (anchor, positive)
                quora_pair_dev = load_dataset("sentence-transformers/quora-duplicates", "pair",
split="train[10000:11000]")
    # (query, answer)
                 natural_questions_dev = load_dataset("sentence-transformers/natural-questions",
split="train[10000:11000]")

    # We can use a dictionary for the evaluation dataset too, but we don't have to. We could also just
use
    # no evaluation dataset, or one dataset.
    eval_dataset = {
        "all-nli-triplet": all_nli_triplet_dev,
        "stsb": stsb_pair_score_dev,
```

```
        "quora": quora_pair_dev,
        "natural-questions": natural_questions_dev,
    }

    # 4. Load several loss functions to train with
    # (anchor, positive), (anchor, positive, negative)
    mnrl_loss = MultipleNegativesRankingLoss(model)
    # (sentence_A, sentence_B) + class
    softmax_loss = SoftmaxLoss(model, model.get_sentence_embedding_dimension(), 3)
    # (sentence_A, sentence_B) + score
    cosent_loss = CoSENTLoss(model)

    # Create a mapping with dataset names to loss functions, so the trainer knows which loss to apply where.
    # Note that you can also just use one loss if all of your training/evaluation datasets use the same loss
    losses = {
        "all-nli-pair": mnrl_loss,
        "all-nli-pair-class": softmax_loss,
        "all-nli-pair-score": cosent_loss,
        "all-nli-triplet": mnrl_loss,
        "stsb": cosent_loss,
        "quora": mnrl_loss,
        "natural-questions": mnrl_loss,
    }

    # 5. Define a simple trainer, although it's recommended to use one with args & evaluators
```

```
    trainer = SentenceTransformerTrainer(
        model=model,
        train_dataset=train_dataset,
        eval_dataset=eval_dataset,
        loss=losses,
    )
    trainer.train()

    # 6. save the trained model and optionally push it to the Hugging Face Hub
    model.save_pretrained("bert-base-all-nli-stsb-quora-nq")
    model.push_to_hub("bert-base-all-nli-stsb-quora-nq")
```

Deprecated Trainingïƒ•

Prior to the Sentence Transformers v3.0 release, models would be trained with
the
[`SentenceTransformer.fit`](../package_reference/sentence_transformer/SentenceTransformer.html#
sentence_transformers.SentenceTransformer.fit
"sentence_transformers.SentenceTransformer.fit") method and a
[`DataLoader`](https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader
"\(in PyTorch v2.5\)") of
[`InputExample`](../package_reference/cross_encoder/cross_encoder.html#sentence_transformers.r
eaders.InputExample
"sentence_transformers.readers.InputExample"), which looked something like
this:

```
from sentence_transformers import SentenceTransformer, InputExample, losses
from torch.utils.data import DataLoader

# Define the model. Either from scratch of by loading a pre-trained model
model = SentenceTransformer("distilbert/distilbert-base-uncased")

# Define your train examples. You need more than just two examples...
train_examples = [
    InputExample(texts=["My first sentence", "My second sentence"], label=0.8),
    InputExample(texts=["Another pair", "Unrelated sentence"], label=0.3),
]

# Define your train dataset, the dataloader and the train loss
train_dataloader = DataLoader(train_examples, shuffle=True, batch_size=16)
train_loss = losses.CosineSimilarityLoss(model)

# Tune the model
model.fit(train_objectives=[(train_dataloader, train_loss)], epochs=1, warmup_steps=100)
```

Since the v3.0 release, using
[`SentenceTransformer.fit`](../package_reference/sentence_transformer/SentenceTransformer.html#
sentence_transformers.SentenceTransformer.fit
"sentence_transformers.SentenceTransformer.fit") is still possible, but it
will initialize a

[`SentenceTransformerTrainer`](../package_reference/sentence_transformer/trainer.html#sentence_transformers.trainer.SentenceTransformerTrainer "sentence_transformers.trainer.SentenceTransformerTrainer") behind the scenes.
It is recommended to use the Trainer directly, as you will have more control
via the
[`SentenceTransformerTrainingArguments`](../package_reference/sentence_transformer/training_args.html#sentence_transformers.training_args.SentenceTransformerTrainingArguments "sentence_transformers.training_args.SentenceTransformerTrainingArguments"),
but existing training scripts relying on
[`SentenceTransformer.fit`](../package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer.fit "sentence_transformers.SentenceTransformer.fit") should still work.

In case there are issues with the updated
[`SentenceTransformer.fit`](../package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer.fit "sentence_transformers.SentenceTransformer.fit"), you can also get exactly the
old behaviour by calling
[`SentenceTransformer.old_fit`](../package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer.old_fit "sentence_transformers.SentenceTransformer.old_fit") instead, but this method
will be deprecated fully in the future.

Best Base Embedding Modelsïƒ•

The quality of your text embedding model depends on which transformer model
you choose. Sadly we cannot infer from a better performance on e.g. the GLUE

or SuperGLUE benchmark that this model will also yield better representations.

To test the suitability of transformer models, I use the [training_nli_v2.py](https://github.com/UKPLab/sentence-transformers/blob/master/examples/training/nli/training_nli_v2.py) script and train on 560k (anchor, positive, negative)-triplets for 1 epoch with batch size 64. I then evaluate on 14 diverse text similarity tasks (clustering, semantic search, duplicate detection etc.) from various domains.

In the following table you find the performance for different models and their performance on this benchmark:

Model	Performance (14 sentence similarity tasks)
[microsoft/mpnet-base](https://huggingface.co/microsoft/mpnet-base)	60.99
[nghuyong/ernie-2.0-en](https://huggingface.co/nghuyong/ernie-2.0-en)	60.73
[microsoft/deberta-base](https://huggingface.co/microsoft/deberta-base)	60.21
[roberta-base](https://huggingface.co/roberta-base)	59.63
[t5-base](https://huggingface.co/t5-base)	59.21
[bert-base-uncased](https://huggingface.co/bert-base-uncased)	59.17
[distilbert-base-uncased](https://huggingface.co/distilbert-base-uncased)	59.03
[nreimers/TinyBERT_L-6_H-768_v2](https://huggingface.co/nreimers/TinyBERT_L-6_H-768_v2)	58.27
[google/t5-v1_1-base](https://huggingface.co/google/t5-v1_1-base)	57.63
[nreimers/MiniLMv2-L6-H768-distilled-from-BERT-Large](https://huggingface.co/nreimers/MiniLMv2-L6-H768-distilled-from-BERT-Large)	57.31
[albert-base-v2](https://huggingface.co/albert-base-v2)	57.14

[microsoft/MiniLM-L12-H384-uncased](https://huggingface.co/microsoft/MiniLM-L12-H384-uncased) | 56.79

[microsoft/deberta-v3-base](https://huggingface.co/microsoft/deberta-v3-base) | 54.46

[Previous](pretrained_models.html "Pretrained Models") [Next](dataset_overview.html "Dataset Overview")

* * *