[![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)

* [Structure of Sentence Transformer Models](../../sentence_transformer/usage/custom_models.html#structure-of-sentence-transformer-models)
* [Sentence Transformer Model from a Transformers Model](../../sentence_transformer/usage/custom_models.html#sentence-transformer-model-from-a-transformers-model)
* [Pretrained Models](../../sentence_transformer/pretrained_models.html)
* [Original Models](../../sentence_transformer/pretrained_models.html#original-models)
* [Semantic Search Models](../../sentence_transformer/pretrained_models.html#semantic-search-models)
* [Multi-QA Models](../../sentence_transformer/pretrained_models.html#multi-qa-models)
* [MSMARCO Passage Models](../../sentence_transformer/pretrained_models.html#msmarco-passage-models)
* [Multilingual Models](../../sentence_transformer/pretrained_models.html#multilingual-models)
* [Semantic Similarity Models](../../sentence_transformer/pretrained_models.html#semantic-similarity-models)
* [Bitext Mining](../../sentence_transformer/pretrained_models.html#bitext-mining)
* [Image & Text-Models](../../sentence_transformer/pretrained_models.html#image-text-models)
* [INSTRUCTOR models](../../sentence_transformer/pretrained_models.html#instructor-models)
* [Scientific Similarity Models](../../sentence_transformer/pretrained_models.html#scientific-similarity-models)
* [Training Overview](../../sentence_transformer/training_overview.html)
* [Why Finetune?](../../sentence_transformer/training_overview.html#why-finetune)
* [Training Components](../../sentence_transformer/training_overview.html#training-components)
* [Dataset](../../sentence_transformer/training_overview.html#dataset)
* [Dataset Format](../../sentence_transformer/training_overview.html#dataset-format)
* [Loss Function](../../sentence_transformer/training_overview.html#loss-function)

* [Training Arguments](../../sentence_transformer/training_overview.html#training-arguments)
* [Evaluator](../../sentence_transformer/training_overview.html#evaluator)
* [Trainer](../../sentence_transformer/training_overview.html#trainer)
* [Callbacks](../../sentence_transformer/training_overview.html#callbacks)
* [Multi-Dataset Training](../../sentence_transformer/training_overview.html#multi-dataset-training)
* [Deprecated Training](../../sentence_transformer/training_overview.html#deprecated-training)
* [Best Base Embedding Models](../../sentence_transformer/training_overview.html#best-base-embedding-models)
* [Dataset Overview](../../sentence_transformer/dataset_overview.html)
* [Datasets on the Hugging Face Hub](../../sentence_transformer/dataset_overview.html#datasets-on-the-hugging-face-hub)
* [Pre-existing Datasets](../../sentence_transformer/dataset_overview.html#pre-existing-datasets)
* [Loss Overview](../../sentence_transformer/loss_overview.html)
* [Loss modifiers](../../sentence_transformer/loss_overview.html#loss-modifiers)
* [Distillation](../../sentence_transformer/loss_overview.html#distillation)
* [Commonly used Loss Functions](../../sentence_transformer/loss_overview.html#commonly-used-loss-functions)
* [Custom Loss Functions](../../sentence_transformer/loss_overview.html#custom-loss-functions)
* [Training Examples](../../sentence_transformer/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#multiplenegativesrankin

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)

* [Training](../../../examples/training/multilingual/README.html#training)
* [Datasets](../../../examples/training/multilingual/README.html#datasets)
* [Sources for Training Data](../../../examples/training/multilingual/README.html#sources-for-training-data)
* [Evaluation](../../../examples/training/multilingual/README.html#evaluation)
* [Available Pre-trained Models](../../../examples/training/multilingual/README.html#available-pre-trained-models)
* [Usage](../../../examples/training/multilingual/README.html#usage)
* [Performance](../../../examples/training/multilingual/README.html#performance)
* [Citation](../../../examples/training/multilingual/README.html#citation)
* [Model Distillation](../../../examples/training/distillation/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](../../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

* [Sentence Transformer](index.html)
  * [SentenceTransformer](SentenceTransformer.html)
    * [SentenceTransformer](SentenceTransformer.html#id1)
    * [SentenceTransformerModelCardData](SentenceTransformer.html#sentencetransformermodelcarddata)
    * [SimilarityFunction](SentenceTransformer.html#similarityfunction)
  * Trainer
    * SentenceTransformerTrainer
  * [Training Arguments](training_args.html)
    * [SentenceTransformerTrainingArguments](training_args.html#sentencetransformertrainingarguments)
  * [Losses](losses.html)
    * [BatchAllTripletLoss](losses.html#batchalltripletloss)
    * [BatchHardSoftMarginTripletLoss](losses.html#batchhardsoftmargintripletloss)
    * [BatchHardTripletLoss](losses.html#batchhardtripletloss)
    * [BatchSemiHardTripletLoss](losses.html#batchsemihardtripletloss)
    * [ContrastiveLoss](losses.html#contrastiveloss)
    * [OnlineContrastiveLoss](losses.html#onlinecontrastiveloss)
    * [ContrastiveTensionLoss](losses.html#contrastivetensionloss)
    * [ContrastiveTensionLossInBatchNegatives](losses.html#contrastivetensionlossinbatchnegatives)
    * [CoSENTLoss](losses.html#cosentloss)
    * [AnglELoss](losses.html#angleloss)
    * [CosineSimilarityLoss](losses.html#cosinesimilarityloss)

    * [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)

      * [ParaphraseMiningEvaluator](evaluation.html#paraphraseminingevaluator)
      * [RerankingEvaluator](evaluation.html#rerankingevaluator)
      * [SentenceEvaluator](evaluation.html#sentenceevaluator)
      * [SequentialEvaluator](evaluation.html#sequentialevaluator)
      * [TranslationEvaluator](evaluation.html#translationevaluator)
      * [TripletEvaluator](evaluation.html#tripletevaluator)
    * [Datasets](datasets.html)
      * [ParallelSentencesDataset](datasets.html#parallelsentencesdataset)
      * [SentenceLabelDataset](datasets.html#sentencelabeldataset)
      * [DenoisingAutoEncoderDataset](datasets.html#denoisingautoencoderdataset)
      * [NoDuplicatesDataLoader](datasets.html#noduplicatesdataloader)
    * [Models](models.html)
      * [Main Classes](models.html#main-classes)
      * [Further Classes](models.html#further-classes)
    * [quantization](quantization.html)
      * [`quantize_embeddings()`](quantization.html#sentence_transformers.quantization.quantize_embeddings)
      * [`semantic_search_faiss()`](quantization.html#sentence_transformers.quantization.semantic_search_faiss)
      * [`semantic_search_usearch()`](quantization.html#sentence_transformers.quantization.semantic_search_usearch)
  * [Cross Encoder](../cross_encoder/index.html)
    * [CrossEncoder](../cross_encoder/cross_encoder.html)
      * [CrossEncoder](../cross_encoder/cross_encoder.html#id1)

* [Training Inputs](../cross_encoder/cross_encoder.html#training-inputs)
* [Evaluation](../cross_encoder/evaluation.html)
* [CEBinaryAccuracyEvaluator](../cross_encoder/evaluation.html#cebinaryaccuracyevaluator)
* [CEBinaryClassificationEvaluator](../cross_encoder/evaluation.html#cebinaryclassificationevaluator)
* [CECorrelationEvaluator](../cross_encoder/evaluation.html#cecorrelationevaluator)
* [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)
* Trainer

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

* * *

Trainerïƒ•

SentenceTransformerTrainerïƒ•

_class _sentence_transformers.trainer.SentenceTransformerTrainer(_model : [SentenceTransformer](SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer") | None = None_, _args : [SentenceTransformerTrainingArguments](training_args.html#sentence_transformers.training_args.SentenceTransformerTrainingArguments "sentence_transformers.training_args.SentenceTransformerTrainingArguments") = None_, _train_dataset : Dataset | DatasetDict | IterableDataset | dict[str, Dataset] | None = None_, _eval_dataset : Dataset | DatasetDict | IterableDataset | dict[str, Dataset] | None = None_, _loss : nn.Module | dict[str, nn.Module] | Callable[[[SentenceTransformer](SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")], [torch.nn.Module](https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module "\(in PyTorch v2.5\)")] | dict[str, Callable[[[SentenceTransformer](SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")], [torch.nn.Module](https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module "\(in PyTorch v2.5\)")]] | None = None_, _evaluator : [SentenceEvaluator](evaluation.html#sentence_transformers.evaluation.SentenceEvaluator "sentence_transformers.evaluation.SentenceEvaluator") | list[[SentenceEvaluator](evaluation.html#sentence_transformers.evaluation.SentenceEvaluator "sentence_transformers.evaluation.SentenceEvaluator")] | None = None_, _data_collator : DataCollator | None = None_, _tokenizer : PreTrainedTokenizerBase | Callable | None = None_, _model_init : Callable[[], [SentenceTransformer](SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")] | None = None_, _compute_metrics : Callable[[EvalPrediction], dict] | None = None_, _callbacks : list[TrainerCallback] | None = None_,

_optimizers : tuple[[torch.optim.Optimizer](https://pytorch.org/docs/stable/optim.html#torch.optim.Optimizer "\(in PyTorch v2.5\)"), [torch.optim.lr_scheduler.LambdaLR](https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.LambdaLR.html#torch.optim.lr_scheduler.LambdaLR "\(in PyTorch v2.5\)")] = (None, None)_, _preprocess_logits_for_metrics : Callable[[[torch.Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)"), [torch.Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")], [torch.Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")] | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\trainer.py#L50-L1233)ïƒ•

SentenceTransformerTrainer is a simple but feature-complete training and eval loop for PyTorch based on the ðŸ¤— Transformers [`Trainer`](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer "\(in transformers vmain\)").

This 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#transfo

rmers.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.

Parameters:

* **model**
([`SentenceTransformer`](SentenceTransformer.html#sentence_transformers.SentenceTransformer

"sentence_transformers.SentenceTransformer"), _optional_) â€“ The model to train, evaluate or use for predictions. If not provided, a model_init must be passed.

* **args** ([`SentenceTransformerTrainingArguments`](training_args.html#sentence_transformers.training_args.SentenceTransformerTrainingArguments "sentence_transformers.training_args.SentenceTransformerTrainingArguments"), _optional_) â€“ The arguments to tweak for training. Will default to a basic instance of [`SentenceTransformerTrainingArguments`](training_args.html#sentence_transformers.training_args.SentenceTransformerTrainingArguments "sentence_transformers.training_args.SentenceTransformerTrainingArguments") with the output_dir set to a directory named _tmp_trainer_ in the current directory if not provided.

* **train_dataset** (Union[[`datasets.Dataset`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset "\(in datasets vmain\)"), [`datasets.DatasetDict`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.DatasetDict "\(in datasets vmain\)"), [`datasets.IterableDataset`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.IterableDataset "\(in datasets vmain\)"), Dict[str, [`datasets.Dataset`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset "\(in datasets vmain\)")]], _optional_) â€“ The dataset to use for training. Must have a format accepted by your loss function, see [Training Overview > Dataset Format](../../../docs/sentence_transformer/training_overview.html#dataset-format).

* **eval_dataset** (Union[[`datasets.Dataset`](https://huggingface.co/docs/datasets/main/en/package_reference/main_

classes#datasets.Dataset "\(in datasets vmain\)"), [`datasets.DatasetDict`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.DatasetDict "\(in datasets vmain\)"), [`datasets.IterableDataset`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.IterableDataset "\(in datasets vmain\)"), Dict[str, [`datasets.Dataset`](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset "\(in datasets vmain\)")]], _optional_) â€“

The dataset to use for evaluation. Must have a format accepted by your loss function, see [Training Overview > Dataset Format](../../../docs/sentence_transformer/training_overview.html#dataset-format).

* **loss** (Optional[Union[[`torch.nn.Module`](https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module "\(in PyTorch v2.5\)"), Dict[str, [`torch.nn.Module`](https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module "\(in PyTorch v2.5\)")], Callable[[[`SentenceTransformer`](SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")], [`torch.nn.Module`](https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module "\(in PyTorch v2.5\)")], Dict[str, Callable[[[`SentenceTransformer`](SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")]]]], _optional_) â€“ The loss function to use for training. Can either be a loss class instance, a dictionary mapping dataset names to loss class instances, a function that returns a loss class instance given a model, or a dictionary mapping dataset names to functions that return a loss class instance given a model. In practice, the latter two

are primarily used for hyper-parameter optimization. Will default to [`CoSENTLoss`](losses.html#sentence_transformers.losses.CoSENTLoss "sentence_transformers.losses.CoSENTLoss") if no `loss` is provided.

* **evaluator** (Union[[`SentenceEvaluator`](evaluation.html#sentence_transformers.evaluation.SentenceEvaluator "sentence_transformers.evaluation.SentenceEvaluator"), List[[`SentenceEvaluator`](evaluation.html#sentence_transformers.evaluation.SentenceEvaluator "sentence_transformers.evaluation.SentenceEvaluator")]], _optional_) â€“ The evaluator instance for useful evaluation metrics during training. You can use an `evaluator` with or without an `eval_dataset`, and vice versa. Generally, the metrics that an `evaluator` returns are more useful than the loss value returned from the `eval_dataset`. A list of evaluators will be wrapped in a [`SequentialEvaluator`](evaluation.html#sentence_transformers.evaluation.SequentialEvaluator "sentence_transformers.evaluation.SequentialEvaluator") to run them sequentially.

* **callbacks** (List of [[`transformers.TrainerCallback`](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback "\(in transformers vmain\)")], _optional_) â€“

A list of callbacks to customize the training loop. Will add those to the list of default callbacks detailed in [here](callback).

If you want to remove one of the default callbacks used, use the [Trainer.remove_callback] method.

* **optimizers** (Tuple[:class:`torch.optim.Optimizer, [`torch.optim.lr_scheduler.LambdaLR`](https://pytorch.org/docs/stable/generated/torch.optim.lr_sche

duler.LambdaLR.html#torch.optim.lr_scheduler.LambdaLR "\(in PyTorch v2.5\)")]`, _optional_ , defaults to (None, None)) â€“ A tuple containing the optimizer and the scheduler to use. Will default to an instance of [`torch.optim.AdamW`](https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html#torch.optim.AdamW "\(in PyTorch v2.5\)") on your model and a scheduler given by [`transformers.get_linear_schedule_with_warmup()`](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.get_linear_schedule_with_warmup "\(in transformers vmain\)") controlled by args.

Important attributes:

> * **model** â€“ Always points to the core model. If using a transformers
> model, it will be a [PreTrainedModel] subclass.
>
> * **model_wrapped** â€“ Always points to the most external model in case
> one or more other modules wrap the original model. This is the model that
> should be used for the forward pass. For example, under DeepSpeed, the inner
> model is wrapped in DeepSpeed and then again in
> torch.nn.DistributedDataParallel. If the inner model hasnâ€™t been wrapped,
> then self.model_wrapped is the same as self.model.
>
> * **is_model_parallel** â€“ Whether or not a model has been switched to a
> model parallel mode (different from data parallelism, this means some of the
> model layers are split on different GPUs).
>
> * **place_model_on_device** â€“ Whether or not to automatically place the
> model on the device - it will be set to False if model parallel or deepspeed

> is used, or if the default TrainingArguments.place_model_on_device is

> overridden to return False .

>

> * **is_in_train** â€“ Whether or not a model is currently running train

> (e.g. when evaluate is called while in train)

>

>

add_callback(_callback_)ïƒ•

Add a callback to the current list of [~transformers.TrainerCallback].

Parameters:

callback (type or [~transformers.TrainerCallback]) â€“ A

[~transformers.TrainerCallback] class or an instance of a

[~transformers.TrainerCallback]. In the first case, will instantiate a member

of that class.

add_model_card_callback(_default_args_dict : dict[str, Any]_) ->

None[[source]](https://github.com/UKPLab/sentence-

transformers/blob/master/sentence_transformers\\trainer.py#L294-L312)ïƒ•

Add a callback responsible for automatically tracking data required for the automatic model card generation

This method is called in the `__init__` method of the `SentenceTransformerTrainer` class.

Parameters:

default_args_dict (_Dict_ _[__str_ _,__Any_ _]_) â€“ A dictionary of the default training arguments, so we can determine which arguments have been changed for the model card.

Note

This method can be overriden by subclassing the trainer to remove/customize this callback in custom uses cases

_static _add_prompts_or_dataset_name_transform(_batch : dict[str, list[Any]]_, _prompts : str | dict[str, str] | None = None_, _prompt_lengths : dict[str, int] | int | None = None_, _dataset_name : str | None = None_, _transform : Callable[[dict[str, list[Any]]], dict[str, list[Any]]] | None = None_, _**kwargs_) -> dict[str, list[Any]][[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\trainer.py#L955-L1017)ï*ƒ•*

A transform/map function that adds prompts or dataset names to the batch.

Parameters:

* **batch** (_dict_ _[__str_ _,__list_ _[__Any_ _]__]_) â€“ The batch of data, where each key is a column name and each value is a list of values.

* **prompts** (_dict_ _[__str_ _,__str_ _]__|__str_ _|__None_ _,__optional_) â€“ An optional mapping of column names to string prompts, or a string prompt for all columns. Defaults to None.

* **prompt_lengths** (_dict_ _[__str_ _,__int_ _]__|__int_ _|__None_ _,__optional_) â€“ An optional mapping of prompts names to prompt token length, or a prompt token length if the prompt is a string. Defaults to None.

* **dataset_name** (_str_ _|__None_ _,__optional_) â€“ The name of this dataset, only if there are multiple datasets that use a different loss. Defaults to None.

* **transform** (_Callable_ _[__[__dict_ _[__str_ _,__list_ _[__Any_ _]__]__]__,__dict_ _[__str_ _,__list_ _[__Any_ _]__]__]__,__optional_) â€“ An optional transform function to apply on the batch before adding prompts, etc. Defaults to None.

Returns:

The â€œjust-in-timeâ€• transformed batch with prompts and/or dataset names

added.

Return type:

dict[str, list[Any]]

compute_loss(_model : [SentenceTransformer](SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")_, _inputs : dict[str, [torch.Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") | Any]_, _return_outputs : bool = False_, _num_items_in_batch =None_) -> [torch.Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") | tuple[[torch.Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)"), dict[str, Any]][[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\trainer.py#L364-L412)ïƒ•

Computes the loss for the SentenceTransformer model.

It uses `self.loss` to compute the loss, which can be a single loss function

or a dictionary of loss functions for different datasets. If the loss is a

dictionary, the dataset name is expected to be passed in the inputs under the key â€œdataset_nameâ€•. This is done automatically in the `add_dataset_name_column` method. Note that even if `return_outputs = True`, the outputs will be empty, as the SentenceTransformers losses do not return outputs.

Parameters:

* **model** ([_SentenceTransformer_](SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")) â€“ The SentenceTransformer model.

* **inputs** (_Dict_ _[__str_ _,__Union_ _[_[_torch.Tensor_](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") _,__Any_ _]__]_) â€“ The input data for the model.

* **return_outputs** (_bool_ _,__optional_) â€“ Whether to return the outputs along with the loss. Defaults to False.

* **num_items_in_batch** (_int_ _,__optional_) â€“ The number of items in the batch. Defaults to None. Unused, but required by the transformers Trainer.

Returns:

The computed loss. If return_outputs is True, returns a tuple of loss and outputs. Otherwise, returns only the loss.

Return type:

Union[[torch.Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)"), Tuple[[torch.Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)"), Dict[str, Any]]]

create_model_card(_language : str | None = None_, _license : str | None = None_, _tags : str | list[str] | None = None_, _model_name : str | None = None_, _finetuned_from : str | None = None_, _tasks : str | list[str] | None = None_, _dataset_tags : str | list[str] | None = None_, _dataset : str | list[str] | None = None_, _dataset_args : str | list[str] | None = None_, _** kwargs_) -> None[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\trainer.py#L1172-L1195)ïƒ•

Creates a draft of a model card using the information available to the Trainer.

Parameters:

* **language** (str, _optional_) â€“ The language of the model (if applicable)

* **license** (str, _optional_) â€“ The license of the model. Will default to the license of the pretrained model used, if the original model given to the Trainer comes from a repo on the Hub.

* **tags** (str or List[str], _optional_) â€“ Some tags to be included in the metadata of the model card.

* **model_name** (str, _optional_) â€“ The name of the model.

* **finetuned_from** (str, _optional_) â€“ The name of the model used to fine-tune this one (if applicable). Will default to the name of the repo of the original model given to the Trainer (if it comes from the Hub).

* **tasks** (str or List[str], _optional_) â€“ One or several task identifiers, to be included in the metadata of the model card.

* **dataset_tags** (str or List[str], _optional_) â€“ One or several dataset tags, to be included in the metadata of the model card.

* **dataset** (str or List[str], _optional_) â€“ One or several dataset identifiers, to be included in the metadata of the model card.

* **dataset_args** (str or List[str], _optional_) â€“ One or several dataset arguments, to be included in the metadata of the model card.

create_optimizer()ïƒ•

Setup the optimizer.

We provide a reasonable default that works well. If you want to use something else, you can pass a tuple in the Trainerâ€™s init through optimizers, or subclass and override this method in a subclass.

create_optimizer_and_scheduler(_num_training_steps : int_)ïƒ•

Setup the optimizer and the learning rate scheduler.

We provide a reasonable default that works well. If you want to use something else, you can pass a tuple in the Trainerâ€™s init through optimizers, or subclass and override this method (or create_optimizer and/or create_scheduler) in a subclass.

create_scheduler(_num_training_steps : int_, _optimizer : [Optimizer](https://pytorch.org/docs/stable/optim.html#torch.optim.Optimizer "\(in PyTorch v2.5\)") | None = None_)ïƒ•

Setup the scheduler. The optimizer of the trainer must have been set up either

before this method is called or passed as an argument.

Parameters:

num_training_steps (_int_) â€“ The number of training steps to do.

evaluate(_eval_dataset : Dataset | dict[str, Dataset] | None = None_, _ignore_keys : list[str] | None = None_, _metric_key_prefix : str = 'eval'_) -> dict[str, float][[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\trainer.py#L449-L461)ïƒ•

Run evaluation and returns metrics.

The calling script will be responsible for providing a method to compute metrics, as they are task-dependent (pass it to the init compute_metrics argument).

You can also subclass and override this method to inject custom behavior.

Parameters:

* **eval_dataset** (Union[Dataset, Dict[str, Dataset]), _optional_) â€“

Pass a dataset if you wish to override self.eval_dataset. If it is a [~datasets.Dataset], columns not accepted by the model.forward() method are automatically removed. If it is a dictionary, it will evaluate on each dataset, prepending the dictionary key to the metric name. Datasets must implement the __len__ method.

<Tip>

If you pass a dictionary with names of datasets as keys and datasets as values, evaluate will run separate evaluations on each dataset. This can be useful to monitor how training affects other datasets or simply to get a more fine-grained evaluation. When used with load_best_model_at_end, make sure metric_for_best_model references exactly one of the datasets. If you, for example, pass in {â€œdata1â€•: data1, â€œdata2â€•: data2} for two datasets data1 and data2, you could specify metric_for_best_model=â€•eval_data1_lossâ€• for using the loss on data1 and metric_for_best_model=â€•eval_data2_lossâ€• for the loss on data2.

</Tip>

* **ignore_keys** (List[str], _optional_) â€“ A list of keys in the output of your model (if it is a dictionary) that should be ignored when gathering predictions.

* **metric_key_prefix** (str, _optional_ , defaults to â€œevalâ€•) â€“ An optional prefix to be used as the metrics key prefix. For example the metrics â€œbleuâ€• will be named â€œeval_bleuâ€• if the prefix is â€œevalâ€• (default)

Returns:

A dictionary containing the evaluation loss and the potential metrics computed
from the predictions. The dictionary also contains the epoch number which
comes from the training state.

get_batch_sampler(_dataset : Dataset_, _batch_size : int_, _drop_last : bool_, _valid_label_columns : list[str] | None = None_, _generator : [Generator](https://pytorch.org/docs/stable/generated/torch.Generator.html#torch.Generator "\(in PyTorch v2.5\)") | None = None_) -> [BatchSampler](https://pytorch.org/docs/stable/data.html#torch.utils.data.BatchSampler "\(in PyTorch v2.5\)") | None[[source]](https://github.com/UKPLab/sentence-transformers\\trainer.py#L549-L603)ïƒ•

Returns the appropriate batch sampler based on the `batch_sampler` argument in
`self.args`. This batch sampler class supports `__len__` and `__iter__`
methods, and is used as the `batch_sampler` to create the
[`torch.utils.data.DataLoader`](https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader "\(in PyTorch v2.5\)").

Note

Override this method to provide a custom batch sampler.

Parameters:

* **dataset** (_Dataset_) â€“ The dataset to sample from.

* **batch_size** (_int_) â€“ Number of samples per batch.

* **drop_last** (_bool_) â€“ If True, drop the last incomplete batch if the dataset size is not divisible by the batch size.

* **valid_label_columns** (_List_ _[__str_ _]_) â€“ List of column names to check for labels. The first column name from `valid_label_columns` found in the dataset will be used as the label column.

* **generator** ([_torch.Generator_](https://pytorch.org/docs/stable/generated/torch.Generator.html#torch.Generator "\(in PyTorch v2.5\)") _,__optional_) â€“ Optional random number generator for shuffling the indices.

get_eval_dataloader(_eval_dataset : Dataset | DatasetDict | IterableDataset | None = None_) -> [DataLoader](https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader "\(in PyTorch v2.5\")[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\trainer.py#L732-L822)ïƒ•

Returns the evaluation [~torch.utils.data.DataLoader].

Subclass and override this method if you want to inject some custom behavior.

Parameters:

eval_dataset (torch.utils.data.Dataset, _optional_) â€“ If provided, will override self.eval_dataset. If it is a [~datasets.Dataset], columns not accepted by the model.forward() method are automatically removed. It must implement __len__.

get_learning_rates()ïƒ•

Returns the learning rate of each parameter from self.optimizer.

get_multi_dataset_batch_sampler(_dataset : [ConcatDataset](https://pytorch.org/docs/stable/data.html#torch.utils.data.ConcatDataset "\(in PyTorch v2.5\")_, _batch_samplers : list[[BatchSampler](https://pytorch.org/docs/stable/data.html#torch.utils.data.BatchSampler "\(in PyTorch v2.5\")]_, _generator : [Generator](https://pytorch.org/docs/stable/generated/torch.Generator.html#torch.Generator "\(in PyTorch v2.5\") | None = None_, _seed : int | None = 0_) -> [BatchSampler](https://pytorch.org/docs/stable/data.html#torch.utils.data.BatchSampler "\(in

PyTorch

v2.5\)")[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\trainer.py#L605-L640)ïƒ•

Returns the appropriate multi-dataset batch sampler based on the

`multi_dataset_batch_sampler` argument in `self.args`. This batch sampler

class supports `__len__` and `__iter__` methods, and is used as the

`batch_sampler` to create the

[`torch.utils.data.DataLoader`](https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader "\(in PyTorch v2.5\)").

Note

Override this method to provide a custom multi-dataset batch sampler.

Parameters:

* **dataset** (_ConcatDataset_) â€“ The concatenation of all datasets.

* **batch_samplers** (_List_ _[__BatchSampler_ _]_) â€“ List of batch samplers for each dataset in the concatenated dataset.

* **generator** ([_torch.Generator_](https://pytorch.org/docs/stable/generated/torch.Generator.html#torch.Generato

r "\(in PyTorch v2.5\)") _,__optional_) â€“ Optional random number generator for shuffling the indices.

* **seed** (_int_ _,__optional_) â€“ Optional seed for the random number generator

get_num_trainable_parameters()ïƒ•

Get the number of trainable parameters.

get_optimizer_group(_param : str | [Parameter](https://pytorch.org/docs/stable/generated/torch.nn.parameter.Parameter.html#torch.nn.parameter.Parameter "\(in PyTorch v2.5\)") | None = None_)ïƒ•

Returns optimizer group for a parameter if given, else returns all optimizer
groups for params.

Parameters:

param (str or torch.nn.parameter.Parameter, _optional_) â€“ The parameter
for which optimizer group needs to be returned.

get_test_dataloader(_test_dataset : Dataset | DatasetDict | IterableDataset_) ->

[DataLoader](https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader "\(in PyTorch v2.5\)")[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\trainer.py#L824-L908)ïƒ•

Returns the training [~torch.utils.data.DataLoader].

Subclass and override this method if you want to inject some custom behavior.

Parameters:

test_dataset (torch.utils.data.Dataset, _optional_) â€“ The test dataset to use. If it is a [~datasets.Dataset], columns not accepted by the model.forward() method are automatically removed. It must implement __len__.

get_train_dataloader() -> [DataLoader](https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader "\(in PyTorch v2.5\)")[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\trainer.py#L642-L730)ïƒ•

Returns the training [~torch.utils.data.DataLoader].

Will use no sampler if train_dataset does not implement __len__, a random

sampler (adapted to distributed training if necessary) otherwise.

Subclass and override this method if you want to inject some custom behavior.

hyperparameter_search(_hp_space : Callable[[optuna.Trial], Dict[str, float]] | None = None_, _compute_objective : Callable[[Dict[str, float]], float] | None = None_, _n_trials : int = 20_, _direction : str | List[str] = 'minimize'_, _backend : str | HPSearchBackend | None = None_, _hp_name : Callable[[optuna.Trial], str] | None = None_, _** kwargs_) -> BestRun | List[BestRun]ïƒ•

Launch an hyperparameter search using optuna or Ray Tune or SigOpt. The
optimized quantity is determined by compute_objective, which defaults to a
function returning the evaluation loss when no metric is provided, the sum of
all metrics otherwise.

<Tip warning={true}>

To use this method, you need to have provided a model_init when initializing
your [Trainer]: we need to reinitialize the model at each new run. This is
incompatible with the optimizers argument, so you need to subclass [Trainer]
and override the method [~Trainer.create_optimizer_and_scheduler] for custom
optimizer/scheduler.

</Tip>

Parameters:

* **hp_space** (Callable[[â€œoptuna.Trialâ€•], Dict[str, float]], _optional_) â€“ A function that defines the hyperparameter search space. Will default to [~trainer_utils.default_hp_space_optuna] or [~trainer_utils.default_hp_space_ray] or [~trainer_utils.default_hp_space_sigopt] depending on your backend.

* **compute_objective** (Callable[[Dict[str, float]], float], _optional_) â€“ A function computing the objective to minimize or maximize from the metrics returned by the evaluate method. Will default to [~trainer_utils.default_compute_objective].

* **n_trials** (int, _optional_ , defaults to 100) â€“ The number of trial runs to test.

* **direction** (str or List[str], _optional_ , defaults to â€œminimizeâ€•) â€“ If itâ€™s single objective optimization, direction is str, can be â€œminimizeâ€• or â€œmaximizeâ€•, you should pick â€œminimizeâ€• when optimizing the validation loss, â€œmaximizeâ€• when optimizing one or several metrics. If itâ€™s multi objectives optimization, direction is List[str], can be List of â€œminimizeâ€• and â€œmaximizeâ€•, you should pick â€œminimizeâ€• when optimizing the validation loss, â€œmaximizeâ€• when optimizing one or several metrics.

* **backend** (str or [~training_utils.HPSearchBackend], _optional_) â€“ The backend to use for hyperparameter search. Will default to optuna or Ray Tune or SigOpt, depending on which one is installed. If all are installed, will default to optuna.

* **hp_name** (Callable[[â€œoptuna.Trialâ€•], str]], _optional_) â€“ A function that defines the trial/run name. Will default to None.

* **kwargs** (Dict[str, Any], _optional_) â€“

Additional keyword arguments passed along to optuna.create_study or ray.tune.run. For more information see:

* the documentation of [optuna.create_study](<https://optuna.readthedocs.io/en/stable/reference/generated/optuna.study.create_study.html>)

* the documentation of [tune.run](<https://docs.ray.io/en/latest/tune/api_docs/execution.html#tune-run>)

* the documentation of [sigopt](<https://app.sigopt.com/docs/endpoints/experiments/create>)

Returns:

All the information about the best run or best runs for multi-objective optimization. Experiment summary can be found in run_summary attribute for Ray backend.

Return type:

[trainer_utils.BestRun or List[trainer_utils.BestRun]]

is_local_process_zero() -> boolïƒ•

Whether or not this process is the local (e.g., on one machine if training in a distributed fashion on several machines) main process.

is_world_process_zero() -> boolïƒ•

Whether or not this process is the global main process (when training in a distributed fashion on several machines, this is only going to be True for one process).

log(_logs : Dict[str, float]_) -> Noneïƒ•

Log logs on the various objects watching training.

Subclass and override this method to inject custom behavior.

Parameters:

logs (Dict[str, float]) â€“ The values to log.

maybe_add_prompts_or_dataset_name_column(_dataset_dict : DatasetDict | Dataset | None_, _prompts : dict[str, dict[str, str]] | dict[str, str] | str | None = None_, _dataset_name : str | None = None_) -> DatasetDict | Dataset | None[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\trainer.py#L1019-L1072)ïƒ•

Maybe add prompts or dataset names to the dataset. We add the dataset_name
column to the dataset if:

1. The loss is a dictionary and the dataset is a DatasetDict, or

2. The prompts contain a mapping to dataset names.

There are 4 cases for the prompts:

1. str: One prompt for all datasets and columns.

2. dict[str, str]: A column to prompt mapping.

3. dict[str, str]: A dataset to prompt mapping.

4. dict[str, dict[str, str]]: A dataset to column to prompt mapping.

And 2 cases for the dataset:

1. Dataset: A single dataset.

2. DatasetDict: A dictionary of datasets.

3A is not allowed, and 2A doesnâ€™t make sense.

Parameters:

dataset_dict (_DatasetDict_ _|__Dataset_ _|__None_) â€“ The dataset to add prompts or dataset names to.

Returns:

The dataset with prompts or dataset names added.

Return type:

DatasetDict | Dataset | None

pop_callback(_callback_)ïƒ•

Remove a callback from the current list of [~transformers.TrainerCallback] and returns it.

If the callback is not found, returns None (and no error is raised).

Parameters:

callback (type or [~transformers.TrainerCallback]) â€“ A [~transformers.TrainerCallback] class or an instance of a [~transformers.TrainerCallback]. In the first case, will pop the first member of that class found in the list of callbacks.

Returns:

The callback removed, if found.

Return type:

[~transformers.TrainerCallback]

propagate_args_to_deepspeed(_auto_find_batch_size =False_)ïƒ•

Sets values in the deepspeed plugin based on the Trainer args

push_to_hub(_commit_message : str | None = 'End of training'_, _blocking : bool = True_, _token : str | None = None_, _** kwargs_) -> strïƒ•

Upload self.model and self.tokenizer to the ðŸ¤— model hub on the repo self.args.hub_model_id.

Parameters:

* **commit_message** (str, _optional_ , defaults to â€œEnd of trainingâ€•) â€“ Message to commit while pushing.

* **blocking** (bool, _optional_ , defaults to True) â€“ Whether the function should return only when the git push has finished.

* **token** (str, _optional_ , defaults to None) â€“ Token with write permission to overwrite

Trainerâ€™s original args.

* **kwargs** (Dict[str, Any], _optional_) â€“ Additional keyword arguments passed along to [~Trainer.create_model_card].

Returns:

The URL of the repository where the model was pushed if blocking=False, or a

Future object tracking the progress of the commit if blocking=True.

remove_callback(_callback_)ïƒ•

Remove a callback from the current list of [~transformers.TrainerCallback].

Parameters:

callback (type or [~transformers.TrainerCallback]) â€“ A

[~transformers.TrainerCallback] class or an instance of a

[~transformers.TrainerCallback]. In the first case, will remove the first

member of that class found in the list of callbacks.

save_model(_output_dir : str | None = None_, __internal_call : bool = False_)ïƒ•

Will save the model, so you can reload it using from_pretrained().

Will only save from the main process.

train(_resume_from_checkpoint : bool | str | None = None_, _trial : optuna.Trial | Dict[str, Any] = None_, _ignore_keys_for_eval : List[str] | None = None_, _** kwargs_)ïƒ•

Main training entry point.

Parameters:

* **resume_from_checkpoint** (str or bool, _optional_) â€“ If a str, local path to a saved checkpoint as saved by a previous instance of [Trainer]. If a bool and equals True, load the last checkpoint in _args.output_dir_ as saved by a previous instance of [Trainer]. If present, training will resume from the model/optimizer/scheduler states loaded here.

* **trial** (optuna.Trial or Dict[str, Any], _optional_) â€“ The trial run or the hyperparameter dictionary for hyperparameter search.

* **ignore_keys_for_eval** (List[str], _optional_) â€“ A list of keys in the output of your model (if it is a dictionary) that should be ignored when gathering predictions for evaluation during the training.

* **kwargs** (Dict[str, Any], _optional_) â€“ Additional keyword arguments used to hide deprecated arguments

[Previous](SentenceTransformer.html "SentenceTransformer") [Next](training_args.html "Training Arguments")

* * *

(C) Copyright 2025.

Built with [Sphinx](https://www.sphinx-doc.org/) using a [theme](https://github.com/readthedocs/sphinx_rtd_theme) provided by [Read the Docs](https://readthedocs.org).