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

Getting Started

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

Sentence Transformer

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

gloss)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

* [Performance

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

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

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

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

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

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

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

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

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

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

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

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

Cross Encoder

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

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

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

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

* [Re-Ranker:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Package Reference

ossinbatchnegatives)

* [CoSENTLoss](../sentence_transformer/losses.html#cosentloss)

* [AnglELoss](../sentence_transformer/losses.html#angleloss)

* [CosineSimilarityLoss](../sentence_transformer/losses.html#cosinesimilarityloss)

* [DenoisingAutoEncoderLoss](../sentence_transformer/losses.html#denoisingautoencoderloss)

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

* [CachedGISTEmbedLoss](../sentence_transformer/losses.html#cachedgistembedloss)

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

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

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

* [Matryoshka2dLoss](../sentence_transformer/losses.html#matryoshka2dloss)

* [AdaptiveLayerLoss](../sentence_transformer/losses.html#adaptivelayerloss)

* [MegaBatchMarginLoss](../sentence_transformer/losses.html#megabatchmarginloss)

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

* [CachedMultipleNegativesRankingLoss](../sentence_transformer/losses.html#cachedmultiplenegativesrankingloss)

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

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

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

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

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

* [BatchSamplers](../sentence_transformer/sampler.html#batchsamplers)
* [MultiDatasetBatchSamplers](../sentence_transformer/sampler.html#multidatasetbatchsamplers)
* [Evaluation](../sentence_transformer/evaluation.html)
* [BinaryClassificationEvaluator](../sentence_transformer/evaluation.html#binaryclassificationevaluator)
* [EmbeddingSimilarityEvaluator](../sentence_transformer/evaluation.html#embeddingsimilarityevaluator)
* [InformationRetrievalEvaluator](../sentence_transformer/evaluation.html#informationretrievalevaluator)
* [NanoBEIREvaluator](../sentence_transformer/evaluation.html#nanobeirevaluator)
* [MSEEvaluator](../sentence_transformer/evaluation.html#mseevaluator)
* [ParaphraseMiningEvaluator](../sentence_transformer/evaluation.html#paraphraseminingevaluator)
* [RerankingEvaluator](../sentence_transformer/evaluation.html#rerankingevaluator)
* [SentenceEvaluator](../sentence_transformer/evaluation.html#sentenceevaluator)
* [SequentialEvaluator](../sentence_transformer/evaluation.html#sequentialevaluator)
* [TranslationEvaluator](../sentence_transformer/evaluation.html#translationevaluator)
* [TripletEvaluator](../sentence_transformer/evaluation.html#tripletevaluator)
* [Datasets](../sentence_transformer/datasets.html)
* [ParallelSentencesDataset](../sentence_transformer/datasets.html#parallelsentencesdataset)
* [SentenceLabelDataset](../sentence_transformer/datasets.html#sentencelabeldataset)
* [DenoisingAutoEncoderDataset](../sentence_transformer/datasets.html#denoisingautoencoderdatas

et)
      * [NoDuplicatesDataLoader](../sentence_transformer/datasets.html#noduplicatesdataloader)
    * [Models](../sentence_transformer/models.html)
      * [Main Classes](../sentence_transformer/models.html#main-classes)
      * [Further Classes](../sentence_transformer/models.html#further-classes)
    * [quantization](../sentence_transformer/quantization.html)
      * [`quantize_embeddings()`](../sentence_transformer/quantization.html#sentence_transformers.quantization.quantize_embeddings)
      * [`semantic_search_faiss()`](../sentence_transformer/quantization.html#sentence_transformers.quantization.semantic_search_faiss)
      * [`semantic_search_usearch()`](../sentence_transformer/quantization.html#sentence_transformers.quantization.semantic_search_usearch)
  * [Cross Encoder](index.html)
    * CrossEncoder
      * CrossEncoder
      * Training Inputs
    * [Evaluation](evaluation.html)
      * [CEBinaryAccuracyEvaluator](evaluation.html#cebinaryaccuracyevaluator)
      * [CEBinaryClassificationEvaluator](evaluation.html#cebinaryclassificationevaluator)
      * [CECorrelationEvaluator](evaluation.html#cecorrelationevaluator)
      * [CEF1Evaluator](evaluation.html#cef1evaluator)
      * [CESoftmaxAccuracyEvaluator](evaluation.html#cesoftmaxaccuracyevaluator)
      * [CERerankingEvaluator](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)
* [Cross Encoder](index.html)
* CrossEncoder
* [Edit on GitHub](https://github.com/UKPLab/sentence-transformers/blob/master/docs/package_reference/cross_encoder/cross_encoder.md)

* * *

CrossEncoderï ƒ•

CrossEncoderï ƒ•

For an introduction to Cross-Encoders, see [Cross-Encoders](../../cross_encoder/usage/usage.html).

_class _sentence_transformers.cross_encoder.CrossEncoder(_model_name : str_, _num_labels : int | None = None_, _max_length : int | None = None_, _device : str | None = None_, _automodel_args : dict | None = None_, _tokenizer_args : dict | None = None_, _config_args : dict | None = None_, _cache_dir : str | None = None_, _trust_remote_code : bool = False_, _revision : str | None = None_, _local_files_only : bool = False_, _default_activation_function =None_, _classifier_dropout : float | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transfor

mers\\cross_encoder\\CrossEncoder.py#L26-L626)ïƒ•

A CrossEncoder takes exactly two sentences / texts as input and either predicts a score or label for this sentence pair. It can for example predict the similarity of the sentence pair on a scale of 0 â€¦ 1.

It does not yield a sentence embedding and does not work for individual sentences.

Parameters:

* **model_name** (_str_) â€“ A model name from Hugging Face Hub that can be loaded with AutoModel, or a path to a local model. We provide several pre-trained CrossEncoder models that can be used for common tasks.

* **num_labels** (_int_ _,__optional_) â€“ Number of labels of the classifier. If 1, the CrossEncoder is a regression model that outputs a continuous score 0â€¦1. If > 1, it output several scores that can be soft-maxed to get probability scores for the different classes. Defaults to None.

* **max_length** (_int_ _,__optional_) â€“ Max length for input sequences. Longer sequences will be truncated. If None, max length of the model will be used. Defaults to None.

* **device** (_str_ _,__optional_) â€“ Device that should be used for the model. If None, it will use CUDA if available. Defaults to None.

* **automodel_args** (_Dict_ _,__optional_) â€“ Arguments passed to AutoModelForSequenceClassification. Defaults to None.

* **tokenizer_args** (_Dict_ _,__optional_) â€“ Arguments passed to AutoTokenizer. Defaults to None.

* **config_args** (_Dict_ _,__optional_) â€“ Arguments passed to AutoConfig. Defaults to None.

* **cache_dir** (str, Path, optional) â€“ Path to the folder where cached files are stored.

* **trust_remote_code** (_bool_ _,__optional_) â€“ Whether or not to allow for custom models defined on the Hub in their own modeling files. This option should only be set to True for repositories you trust and in which you have read the code, as it will execute code present on the Hub on your local machine. Defaults to False.

* **revision** (_Optional_ _[__str_ _]__,__optional_) â€“ The specific model version to use. It can be a branch name, a tag name, or a commit id, for a stored model on Hugging Face. Defaults to None.

* **local_files_only** (_bool_ _,__optional_) â€“ If True, avoid downloading the model. Defaults to False.

* **default_activation_function** (_Callable_ _,__optional_) â€“ Callable (like nn.Sigmoid) about the default activation function that should be used on-top of model.predict(). If None. nn.Sigmoid() will be used if num_labels=1, else nn.Identity(). Defaults to None.

* **classifier_dropout** (_float_ _,__optional_) â€“ The dropout ratio for the classification head. Defaults to None.

fit(_train_dataloader: ~torch.utils.data.dataloader.DataLoader, evaluator: ~sentence_transformers.evaluation.SentenceEvaluator.SentenceEvaluator | None = None, epochs: int = 1, loss_fct=None, activation_fct=Identity(), scheduler: str = 'WarmupLinear', warmup_steps: int = 10000, optimizer_class: type[~torch.optim.optimizer.Optimizer] = <class 'torch.optim.adamw.AdamW'>, optimizer_params: dict[str, object] = {'lr': 2e-05}, weight_decay: float = 0.01, evaluation_steps: int = 0, output_path: str | None = None, save_best_model: bool = True, max_grad_norm: float = 1, use_amp: bool = False, callback: ~typing.Callable[[float, int, int], None] | None = None, show_progress_bar: bool = True_) -> None[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\cross_encoder\\CrossEncoder.py#L180-L314)ïƒ•

Train the model with the given training objective Each training objective is
sampled in turn for one batch. We sample only as many batches from each
objective as there are in the smallest one to make sure of equal training with
each dataset.

Parameters:

* **train_dataloader** (_DataLoader_) â€“ DataLoader with training InputExamples

* **evaluator**

([_SentenceEvaluator_](../sentence_transformer/evaluation.html#sentence_transformers.evaluation.SentenceEvaluator "sentence_transformers.evaluation.SentenceEvaluator") _,__optional_) â€“ An evaluator (sentence_transformers.evaluation) evaluates the model performance during training on held-out dev data. It is used to determine the best model that is saved to disc. Defaults to None.

* **epochs** (_int_ _,__optional_) â€“ Number of epochs for training. Defaults to 1.

* **loss_fct** â€“ Which loss function to use for training. If None, will use nn.BCEWithLogitsLoss() if self.config.num_labels == 1 else nn.CrossEntropyLoss(). Defaults to None.

* **activation_fct** â€“ Activation function applied on top of logits output of model.

* **scheduler** (_str_ _,__optional_) â€“ Learning rate scheduler. Available schedulers: constantlr, warmupconstant, warmuplinear, warmupcosine, warmupcosinewithhardrestarts. Defaults to â€œWarmupLinearâ€•.

* **warmup_steps** (_int_ _,__optional_) â€“ Behavior depends on the scheduler. For WarmupLinear (default), the learning rate is increased from o up to the maximal learning rate. After these many training steps, the learning rate is decreased linearly back to zero. Defaults to 10000.

* **optimizer_class** (_Type_ _[__Optimizer_ _]__,__optional_) â€“ Optimizer. Defaults to torch.optim.AdamW.

* **optimizer_params** (_Dict_ _[__str_ _,__object_ _]__,__optional_) â€“ Optimizer parameters. Defaults to {â€œlrâ€•: 2e-5}.

* **weight_decay** (_float_ _,__optional_) â€“ Weight decay for model parameters. Defaults to

0.01.

* **evaluation_steps** (_int_ _,__optional_) â€“ If > 0, evaluate the model using evaluator after each number of training steps. Defaults to 0.

* **output_path** (_str_ _,__optional_) â€“ Storage path for the model and evaluation files. Defaults to None.

* **save_best_model** (_bool_ _,__optional_) â€“ If true, the best model (according to evaluator) is stored at output_path. Defaults to True.

* **max_grad_norm** (_float_ _,__optional_) â€“ Used for gradient normalization. Defaults to 1.

* **use_amp** (_bool_ _,__optional_) â€“ Use Automatic Mixed Precision (AMP). Only for Pytorch >= 1.6.0. Defaults to False.

* **callback** (_Callable_ _[__[__float_ _,__int_ _,__int_ _]__,__None_ _]__,__optional_) â€“ Callback function that is invoked after each evaluation. It must accept the following three parameters in this order: score, epoch, steps. Defaults to None.

* **show_progress_bar** (_bool_ _,__optional_) â€“ If True, output a tqdm progress bar. Defaults to True.

predict(_sentences : tuple[str, str] | list[str]_, _batch_size : int = 32_, _show_progress_bar : bool | None = None_, _num_workers : int = 0_, _activation_fct : Callable | None = None_, _apply_softmax : bool | None = False_, _convert_to_numpy : Literal[False] = True_, _convert_to_tensor : Literal[False] = False_) -> [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in

PyTorch

v2.5\")[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\cross_encoder\\CrossEncoder.py#L368-L459)ïƒ•

predict(_sentences : list[tuple[str, str]] | list[list[str]] | tuple[str, str] | list[str]_, _batch_size : int = 32_, _show_progress_bar : bool | None = None_, _num_workers : int = 0_, _activation_fct : Callable | None = None_, _apply_softmax : bool | None = False_, _convert_to_numpy : Literal[True] = True_, _convert_to_tensor : Literal[False] = False_) -> ndarray

predict(_sentences : list[tuple[str, str]] | list[list[str]] | tuple[str, str] | list[str]_, _batch_size : int = 32_, _show_progress_bar : bool | None = None_, _num_workers : int = 0_, _activation_fct : Callable | None = None_, _apply_softmax : bool | None = False_, _convert_to_numpy : bool = True_, _convert_to_tensor : Literal[True] = False_) -> [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\")

predict(_sentences : list[tuple[str, str]] | list[list[str]]_, _batch_size : int = 32_, _show_progress_bar : bool | None = None_, _num_workers : int = 0_, _activation_fct : Callable | None = None_, _apply_softmax : bool | None = False_, _convert_to_numpy : Literal[False] = True_, _convert_to_tensor : Literal[False] = False_) -> list[[Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\")]

Performs predictions with the CrossEncoder on the given sentence pairs.

Parameters:

* **sentences** (_Union_ _[__List_ _[__Tuple_ _[__str_ _,__str_ _]__]__,__Tuple_ _[__str_ _,__str_ _]__]_) â€“ A list of sentence pairs [(Sent1, Sent2), (Sent3, Sent4)] or one sentence pair

(Sent1, Sent2).

* **batch_size** (_int_ _,__optional_) â€“ Batch size for encoding. Defaults to 32.

* **show_progress_bar** (_bool_ _,__optional_) â€“ Output progress bar. Defaults to None.

* **num_workers** (_int_ _,__optional_) â€“ Number of workers for tokenization. Defaults to 0.

* **activation_fct** (_callable_ _,__optional_) â€“ Activation function applied on the logits output of the CrossEncoder. If None, nn.Sigmoid() will be used if num_labels=1, else nn.Identity. Defaults to None.

* **convert_to_numpy** (_bool_ _,__optional_) â€“ Convert the output to a numpy matrix. Defaults to True.

* **apply_softmax** (_bool_ _,__optional_) â€“ If there are more than 2 dimensions and apply_softmax=True, applies softmax on the logits output. Defaults to False.

* **convert_to_tensor** (_bool_ _,__optional_) â€“ Convert the output to a tensor. Defaults to False.

Returns:

Predictions for the passed sentence pairs. The return type depends on the convert_to_numpy and convert_to_tensor parameters. If convert_to_tensor is

True, the output will be a torch.Tensor. If convert_to_numpy is True, the output will be a numpy.ndarray. Otherwise, the output will be a list of float values.

Return type:

Union[List[float], np.ndarray, [torch.Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")]

Examples

```
from sentence_transformers import CrossEncoder

model = CrossEncoder("cross-encoder/stsb-roberta-base")
sentences = [["I love cats", "Cats are amazing"], ["I prefer dogs", "Dogs are loyal"]]
model.predict(sentences)
# => array([0.6912767, 0.4303499], dtype=float32)
```

push_to_hub(_repo_id : str_, _use_temp_dir : bool | None = None_, _commit_message : str | None = None_, _private : bool | None = None_, _token : bool | str | None = None_, _max_shard_size : int | str | None = '5GB'_, _create_pr : bool = False_, _safe_serialization : bool = True_, _revision : str =

None_, _commit_description : str = None_, _tags : List[str] | None = None_, _** deprecated_kwargs_) -> strï*ƒ*•

Upload the {object_files} to the ðŸ¤— Model Hub.

Parameters:

* **repo_id** (str) â€“ The name of the repository you want to push your {object} to. It should contain your organization name when pushing to a given organization.

* **use_temp_dir** (bool, _optional_) â€“ Whether or not to use a temporary directory to store the files saved before they are pushed to the Hub. Will default to True if there is no directory named like repo_id, False otherwise.

* **commit_message** (str, _optional_) â€“ Message to commit while pushing. Will default to â€œUpload {object}â€•.

* **private** (bool, _optional_) â€“ Whether or not the repository created should be private.

* **token** (bool or str, _optional_) â€“ The token to use as HTTP bearer authorization for remote files. If True, will use the token generated when running huggingface-cli login (stored in ~/.huggingface). Will default to True if repo_url is not specified.

* **max_shard_size** (int or str, _optional_ , defaults to â€œ5GBâ€•) â€“ Only applicable for

models. The maximum size for a checkpoint before being sharded. Checkpoints shard will then be each of size lower than this size. If expressed as a string, needs to be digits followed by a unit (like â€œ5MBâ€•). We default it to â€œ5GBâ€• so that users can easily load models on free-tier Google Colab instances without any CPU OOM issues.

* **create_pr** (bool, _optional_ , defaults to False) â€“ Whether or not to create a PR with the uploaded files or directly commit.

* **safe_serialization** (bool, _optional_ , defaults to True) â€“ Whether or not to convert the model weights in safetensors format for safer serialization.

* **revision** (str, _optional_) â€“ Branch to push the uploaded files to.

* **commit_description** (str, _optional_) â€“ The description of the commit that will be created

* **tags** (List[str], _optional_) â€“ List of tags to push on the Hub.

Examples:

```python from transformers import {object_class}

{object} = {object_class}.from_pretrained(â€œgoogle-bert/bert-base-casedâ€•)

# Push the {object} to your namespace with the name â€œmy-finetuned-bertâ€•.
{object}.push_to_hub(â€œmy-finetuned-bertâ€•)

# Push the {object} to an organization with the name â€œmy-finetuned-bertâ€•.
```

{object}.push_to_hub(â€œhuggingface/my-finetuned-bertâ€•) ```

rank(_query : str_, _documents : list[str]_, _top_k : int | None = None_, _return_documents : bool = False_, _batch_size : int = 32_, _show_progress_bar : bool | None = None_, _num_workers : int = 0_, _activation_fct =None_, _apply_softmax =False_, _convert_to_numpy : bool = True_, _convert_to_tensor : bool = False_) -> list[dict[Literal['corpus_id', 'score', 'text'], int | float | str]][[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\cross_encoder\\CrossEncoder.py#L461-L554)ïƒ•

Performs ranking with the CrossEncoder on the given query and documents.

Returns a sorted list with the document indices and scores.

Parameters:

* **query** (_str_) â€“ A single query.

* **documents** (_List_ _[__str_ _]_) â€“ A list of documents.

* **top_k** (_Optional_ _[__int_ _]__,__optional_) â€“ Return the top-k documents. If None, all documents are returned. Defaults to None.

* **return_documents** (_bool_ _,__optional_) â€“ If True, also returns the documents. If False, only returns the indices and scores. Defaults to False.

* **batch_size** (_int_ _,__optional_) â€“ Batch size for encoding. Defaults to 32.

* **show_progress_bar** (_bool_ _,__optional_) â€“ Output progress bar. Defaults to None.

* **num_workers** (_int_ _,__optional_) â€“ Number of workers for tokenization. Defaults to 0.

* **activation_fct** (_[__type_ _]__,__optional_) â€“ Activation function applied on the logits output of the CrossEncoder. If None, nn.Sigmoid() will be used if num_labels=1, else nn.Identity. Defaults to None.

* **convert_to_numpy** (_bool_ _,__optional_) â€“ Convert the output to a numpy matrix. Defaults to True.

* **apply_softmax** (_bool_ _,__optional_) â€“ If there are more than 2 dimensions and apply_softmax=True, applies softmax on the logits output. Defaults to False.

* **convert_to_tensor** (_bool_ _,__optional_) â€“ Convert the output to a tensor. Defaults to False.

Returns:

A sorted list with the â€œcorpus_idâ€•, â€œscoreâ€•, and optionally â€œtextâ€•
of the documents.

Return type:

List[Dict[Literal[â€œcorpus_idâ€•, â€œscoreâ€•, â€œtextâ€•], Union[int, float,

str]]]

Example

```
from sentence_transformers import CrossEncoder
model = CrossEncoder("cross-encoder/ms-marco-MiniLM-L-6-v2")

query = "Who wrote 'To Kill a Mockingbird'?"
documents = [
    "'To Kill a Mockingbird' is a novel by Harper Lee published in 1960. It was immediately successful, winning the Pulitzer Prize, and has become a classic of modern American literature.",
    "The novel 'Moby-Dick' was written by Herman Melville and first published in 1851. It is considered a masterpiece of American literature and deals with complex themes of obsession, revenge, and the conflict between good and evil.",
    "Harper Lee, an American novelist widely known for her novel 'To Kill a Mockingbird', was born in 1926 in Monroeville, Alabama. She received the Pulitzer Prize for Fiction in 1961.",
    "Jane Austen was an English novelist known primarily for her six major novels, which interpret, critique and comment upon the British landed gentry at the end of the 18th century.",
    "The 'Harry Potter' series, which consists of seven fantasy novels written by British author J.K. Rowling, is among the most popular and critically acclaimed books of the modern era.",
    "'The Great Gatsby', a novel written by American author F. Scott Fitzgerald, was published in
```

1925. The story is set in the Jazz Age and follows the life of millionaire Jay Gatsby and his pursuit of Daisy Buchanan."

]

model.rank(query, documents, return_documents=True)

[{'corpus_id': 0,

'score': 10.67858,

'text': "'To Kill a Mockingbird' is a novel by Harper Lee published in 1960. It was immediately successful, winning the Pulitzer Prize, and has become a classic of modern American literature."},

{'corpus_id': 2,

'score': 9.761677,

'text': "Harper Lee, an American novelist widely known for her novel 'To Kill a Mockingbird', was born in 1926 in Monroeville, Alabama. She received the Pulitzer Prize for Fiction in 1961."},

{'corpus_id': 1,

'score': -3.3099542,

'text': "The novel 'Moby-Dick' was written by Herman Melville and first published in 1851. It is considered a masterpiece of American literature and deals with complex themes of obsession, revenge, and the conflict between good and evil."},

{'corpus_id': 5,

'score': -4.8989105,

'text': "'The Great Gatsby', a novel written by American author F. Scott Fitzgerald, was published in 1925. The story is set in the Jazz Age and follows the life of millionaire Jay Gatsby and his pursuit of Daisy Buchanan."},

{'corpus_id': 4,

'score': -5.082967,

'text': "The 'Harry Potter' series, which consists of seven fantasy novels written by British author J.K. Rowling, is among the most popular and critically acclaimed books of the modern era."}]

save(_path : str_, _*_ , _safe_serialization : bool = True_, _** kwargs_) -> None[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\cross_encoder\\CrossEncoder.py#L567-L576)ïƒ•

Saves the model and tokenizer to path; identical to save_pretrained

save_pretrained(_path : str_, _*_ , _safe_serialization : bool = True_, _** kwargs_) -> None[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\cross_encoder\\CrossEncoder.py#L578-L582)ïƒ•

Saves the model and tokenizer to path; identical to save

Training Inputsïƒ•

_class _sentence_transformers.readers.InputExample(_guid : str = ''_, _texts : list[str] | None = None_, _label : int | float = 0_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\readers\\InputExample.py#L14-L31)ïƒ•

Structure for one input example with texts, the label and a unique id

Creates one InputExample with the given texts, guid and label

Parameters:

* **guid** â€“ id for the example

* **texts** â€“ the texts for the example.

* **label** â€“ the label for the example

[Previous](index.html "Cross Encoder") [Next](evaluation.html "Evaluation")

* * *