**BERT**

**Overview**

The BERT model was proposed in [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](https://arxiv.org/abs/1810.04805) by Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. It’s a bidirectional transformer pretrained using a combination of masked language modeling objective and next sentence prediction on a large corpus comprising the Toronto Book Corpus and Wikipedia.

The abstract from the paper is the following:

*We introduce a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models, BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.*

*BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).*

This model was contributed by [thomwolf](https://huggingface.co/thomwolf). The original code can be found [here](https://github.com/google-research/bert).

**Usage tips**

* BERT is a model with absolute position embeddings so it’s usually advised to pad the inputs on the right rather than the left.
* BERT was trained with the masked language modeling (MLM) and next sentence prediction (NSP) objectives. It is efficient at predicting masked tokens and at NLU in general, but is not optimal for text generation.
* Corrupts the inputs by using random masking, more precisely, during pretraining, a given percentage of tokens (usually 15%) is masked by:
  + a special mask token with probability 0.8
  + a random token different from the one masked with probability 0.1
  + the same token with probability 0.1
* The model must predict the original sentence, but has a second objective: inputs are two sentences A and B (with a separation token in between). With probability 50%, the sentences are consecutive in the corpus, in the remaining 50% they are not related. The model has to predict if the sentences are consecutive or not.

**Resources**

A list of official Hugging Face and community (indicated by 🌎) resources to help you get started with BERT. If you’re interested in submitting a resource to be included here, please feel free to open a Pull Request and we’ll review it! The resource should ideally demonstrate something new instead of duplicating an existing resource.

Text Classification

* A blog post on [BERT Text Classification in a different language](https://www.philschmid.de/bert-text-classification-in-a-different-language).
* A notebook for [Finetuning BERT (and friends) for multi-label text classification](https://colab.research.google.com/github/NielsRogge/Transformers-Tutorials/blob/master/BERT/Fine_tuning_BERT_(and_friends)_for_multi_label_text_classification.ipynb).
* A notebook on how to [Finetune BERT for multi-label classification using PyTorch](https://colab.research.google.com/github/abhimishra91/transformers-tutorials/blob/master/transformers_multi_label_classification.ipynb). 🌎
* A notebook on how to [warm-start an EncoderDecoder model with BERT for summarization](https://colab.research.google.com/github/patrickvonplaten/notebooks/blob/master/BERT2BERT_for_CNN_Dailymail.ipynb).
* [BertForSequenceClassification](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForSequenceClassification) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/pytorch/text-classification) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/text_classification.ipynb).
* [TFBertForSequenceClassification](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForSequenceClassification) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/tensorflow/text-classification) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/text_classification-tf.ipynb).
* [FlaxBertForSequenceClassification](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.FlaxBertForSequenceClassification) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/flax/text-classification) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/text_classification_flax.ipynb).
* [Text classification task guide](https://huggingface.co/docs/transformers/main/tasks/sequence_classification)

Token Classification

* A blog post on how to use [Hugging Face Transformers with Keras: Fine-tune a non-English BERT for Named Entity Recognition](https://www.philschmid.de/huggingface-transformers-keras-tf).
* A notebook for [Finetuning BERT for named-entity recognition](https://colab.research.google.com/github/NielsRogge/Transformers-Tutorials/blob/master/Custom_Named_Entity_Recognition_with_BERT_only_first_wordpiece.ipynb) using only the first wordpiece of each word in the word label during tokenization. To propagate the label of the word to all wordpieces, see this [version](https://github.com/NielsRogge/Transformers-Tutorials/blob/master/BERT/Custom_Named_Entity_Recognition_with_BERT.ipynb) of the notebook instead.
* [BertForTokenClassification](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForTokenClassification) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/pytorch/token-classification) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/token_classification.ipynb).
* [TFBertForTokenClassification](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForTokenClassification) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/tensorflow/token-classification) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/token_classification-tf.ipynb).
* [FlaxBertForTokenClassification](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.FlaxBertForTokenClassification) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/flax/token-classification).
* [Token classification](https://huggingface.co/course/chapter7/2?fw=pt) chapter of the 🤗 Hugging Face Course.
* [Token classification task guide](https://huggingface.co/docs/transformers/main/tasks/token_classification)

Fill-Mask

* [BertForMaskedLM](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForMaskedLM) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/pytorch/language-modeling#robertabertdistilbert-and-masked-language-modeling) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/language_modeling.ipynb).
* [TFBertForMaskedLM](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForMaskedLM) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/tensorflow/language-modeling#run_mlmpy) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/language_modeling-tf.ipynb).
* [FlaxBertForMaskedLM](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.FlaxBertForMaskedLM) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/flax/language-modeling#masked-language-modeling) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/masked_language_modeling_flax.ipynb).
* [Masked language modeling](https://huggingface.co/course/chapter7/3?fw=pt) chapter of the 🤗 Hugging Face Course.
* [Masked language modeling task guide](https://huggingface.co/docs/transformers/main/tasks/masked_language_modeling)

Question Answering

* [BertForQuestionAnswering](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForQuestionAnswering) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/pytorch/question-answering) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/question_answering.ipynb).
* [TFBertForQuestionAnswering](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForQuestionAnswering) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/tensorflow/question-answering) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/question_answering-tf.ipynb).
* [FlaxBertForQuestionAnswering](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.FlaxBertForQuestionAnswering) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/flax/question-answering).
* [Question answering](https://huggingface.co/course/chapter7/7?fw=pt) chapter of the 🤗 Hugging Face Course.
* [Question answering task guide](https://huggingface.co/docs/transformers/main/tasks/question_answering)

**Multiple choice**

* [BertForMultipleChoice](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForMultipleChoice) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/pytorch/multiple-choice) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/multiple_choice.ipynb).
* [TFBertForMultipleChoice](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForMultipleChoice) is supported by this [example script](https://github.com/huggingface/transformers/tree/main/examples/tensorflow/multiple-choice) and [notebook](https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/multiple_choice-tf.ipynb).
* [Multiple choice task guide](https://huggingface.co/docs/transformers/main/tasks/multiple_choice)

⚡️ **Inference**

* A blog post on how to [Accelerate BERT inference with Hugging Face Transformers and AWS Inferentia](https://huggingface.co/blog/bert-inferentia-sagemaker).
* A blog post on how to [Accelerate BERT inference with DeepSpeed-Inference on GPUs](https://www.philschmid.de/bert-deepspeed-inference).

⚙️ **Pretraining**

* A blog post on [Pre-Training BERT with Hugging Face Transformers and Habana Gaudi](https://www.philschmid.de/pre-training-bert-habana).

🚀 **Deploy**

* A blog post on how to [Convert Transformers to ONNX with Hugging Face Optimum](https://www.philschmid.de/convert-transformers-to-onnx).
* A blog post on how to [Setup Deep Learning environment for Hugging Face Transformers with Habana Gaudi on AWS](https://www.philschmid.de/getting-started-habana-gaudi#conclusion).
* A blog post on [Autoscaling BERT with Hugging Face Transformers, Amazon SageMaker and Terraform module](https://www.philschmid.de/terraform-huggingface-amazon-sagemaker-advanced).
* A blog post on [Serverless BERT with HuggingFace, AWS Lambda, and Docker](https://www.philschmid.de/serverless-bert-with-huggingface-aws-lambda-docker).
* A blog post on [Hugging Face Transformers BERT fine-tuning using Amazon SageMaker and Training Compiler](https://www.philschmid.de/huggingface-amazon-sagemaker-training-compiler).
* A blog post on [Task-specific knowledge distillation for BERT using Transformers & Amazon SageMaker](https://www.philschmid.de/knowledge-distillation-bert-transformers).

**BertConfig**

classtransformers.**BertConfig**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/configuration_bert.py#L72)

( vocab\_size = 30522hidden\_size = 768num\_hidden\_layers = 12num\_attention\_heads = 12intermediate\_size = 3072hidden\_act = 'gelu'hidden\_dropout\_prob = 0.1attention\_probs\_dropout\_prob = 0.1max\_position\_embeddings = 512type\_vocab\_size = 2initializer\_range = 0.02layer\_norm\_eps = 1e-12pad\_token\_id = 0position\_embedding\_type = 'absolute'use\_cache = Trueclassifier\_dropout = None\*\*kwargs )

Expand 16 parameters

**Parameters**

* **vocab\_size** (int, *optional*, defaults to 30522) — Vocabulary size of the BERT model. Defines the number of different tokens that can be represented by the inputs\_ids passed when calling [BertModel](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertModel) or [TFBertModel](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertModel).
* **hidden\_size** (int, *optional*, defaults to 768) — Dimensionality of the encoder layers and the pooler layer.
* **num\_hidden\_layers** (int, *optional*, defaults to 12) — Number of hidden layers in the Transformer encoder.
* **num\_attention\_heads** (int, *optional*, defaults to 12) — Number of attention heads for each attention layer in the Transformer encoder.
* **intermediate\_size** (int, *optional*, defaults to 3072) — Dimensionality of the “intermediate” (often named feed-forward) layer in the Transformer encoder.
* **hidden\_act** (str or Callable, *optional*, defaults to "gelu") — The non-linear activation function (function or string) in the encoder and pooler. If string, "gelu", "relu", "silu" and "gelu\_new" are supported.
* **hidden\_dropout\_prob** (float, *optional*, defaults to 0.1) — The dropout probability for all fully connected layers in the embeddings, encoder, and pooler.
* **attention\_probs\_dropout\_prob** (float, *optional*, defaults to 0.1) — The dropout ratio for the attention probabilities.
* **max\_position\_embeddings** (int, *optional*, defaults to 512) — The maximum sequence length that this model might ever be used with. Typically set this to something large just in case (e.g., 512 or 1024 or 2048).
* **type\_vocab\_size** (int, *optional*, defaults to 2) — The vocabulary size of the token\_type\_ids passed when calling [BertModel](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertModel) or [TFBertModel](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertModel).
* **initializer\_range** (float, *optional*, defaults to 0.02) — The standard deviation of the truncated\_normal\_initializer for initializing all weight matrices.
* **layer\_norm\_eps** (float, *optional*, defaults to 1e-12) — The epsilon used by the layer normalization layers.
* **position\_embedding\_type** (str, *optional*, defaults to "absolute") — Type of position embedding. Choose one of "absolute", "relative\_key", "relative\_key\_query". For positional embeddings use "absolute". For more information on "relative\_key", please refer to [Self-Attention with Relative Position Representations (Shaw et al.)](https://arxiv.org/abs/1803.02155). For more information on "relative\_key\_query", please refer to *Method 4* in [Improve Transformer Models with Better Relative Position Embeddings (Huang et al.)](https://arxiv.org/abs/2009.13658).
* **is\_decoder** (bool, *optional*, defaults to False) — Whether the model is used as a decoder or not. If False, the model is used as an encoder.
* **use\_cache** (bool, *optional*, defaults to True) — Whether or not the model should return the last key/values attentions (not used by all models). Only relevant if config.is\_decoder=True.
* **classifier\_dropout** (float, *optional*) — The dropout ratio for the classification head.

This is the configuration class to store the configuration of a [BertModel](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertModel) or a [TFBertModel](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertModel). It is used to instantiate a BERT model according to the specified arguments, defining the model architecture. Instantiating a configuration with the defaults will yield a similar configuration to that of the BERT [bert-base-uncased](https://huggingface.co/bert-base-uncased) architecture.

Configuration objects inherit from [PretrainedConfig](https://huggingface.co/docs/transformers/main/en/main_classes/configuration#transformers.PretrainedConfig) and can be used to control the model outputs. Read the documentation from [PretrainedConfig](https://huggingface.co/docs/transformers/main/en/main_classes/configuration#transformers.PretrainedConfig) for more information.

Examples:

Copied

>>> from transformers import BertConfig, BertModel

>>> *# Initializing a BERT bert-base-uncased style configuration*

>>> configuration = BertConfig()

>>> *# Initializing a model (with random weights) from the bert-base-uncased style configuration*

>>> model = BertModel(configuration)

>>> *# Accessing the model configuration*

>>> configuration = model.config

**BertTokenizer**

classtransformers.**BertTokenizer**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/tokenization_bert.py#L137)

( vocab\_filedo\_lower\_case = Truedo\_basic\_tokenize = Truenever\_split = Noneunk\_token = '[UNK]'sep\_token = '[SEP]'pad\_token = '[PAD]'cls\_token = '[CLS]'mask\_token = '[MASK]'tokenize\_chinese\_chars = Truestrip\_accents = None\*\*kwargs )

Expand 11 parameters

**Parameters**

* **vocab\_file** (str) — File containing the vocabulary.
* **do\_lower\_case** (bool, *optional*, defaults to True) — Whether or not to lowercase the input when tokenizing.
* **do\_basic\_tokenize** (bool, *optional*, defaults to True) — Whether or not to do basic tokenization before WordPiece.
* **never\_split** (Iterable, *optional*) — Collection of tokens which will never be split during tokenization. Only has an effect when do\_basic\_tokenize=True
* **unk\_token** (str, *optional*, defaults to "[UNK]") — The unknown token. A token that is not in the vocabulary cannot be converted to an ID and is set to be this token instead.
* **sep\_token** (str, *optional*, defaults to "[SEP]") — The separator token, which is used when building a sequence from multiple sequences, e.g. two sequences for sequence classification or for a text and a question for question answering. It is also used as the last token of a sequence built with special tokens.
* **pad\_token** (str, *optional*, defaults to "[PAD]") — The token used for padding, for example when batching sequences of different lengths.
* **cls\_token** (str, *optional*, defaults to "[CLS]") — The classifier token which is used when doing sequence classification (classification of the whole sequence instead of per-token classification). It is the first token of the sequence when built with special tokens.
* **mask\_token** (str, *optional*, defaults to "[MASK]") — The token used for masking values. This is the token used when training this model with masked language modeling. This is the token which the model will try to predict.
* **tokenize\_chinese\_chars** (bool, *optional*, defaults to True) — Whether or not to tokenize Chinese characters.

This should likely be deactivated for Japanese (see this [issue](https://github.com/huggingface/transformers/issues/328)).

* **strip\_accents** (bool, *optional*) — Whether or not to strip all accents. If this option is not specified, then it will be determined by the value for lowercase (as in the original BERT).

Construct a BERT tokenizer. Based on WordPiece.

This tokenizer inherits from [PreTrainedTokenizer](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizer) which contains most of the main methods. Users should refer to this superclass for more information regarding those methods.

build\_inputs\_with\_special\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/tokenization_bert.py#L270)

( token\_ids\_0: typing.List[int]token\_ids\_1: typing.Optional[typing.List[int]] = None ) **→** List[int]

**Parameters**

* **token\_ids\_0** (List[int]) — List of IDs to which the special tokens will be added.
* **token\_ids\_1** (List[int], *optional*) — Optional second list of IDs for sequence pairs.

**Returns**

List[int]

List of [input IDs](https://huggingface.co/docs/transformers/main/glossary#input-ids) with the appropriate special tokens.

Build model inputs from a sequence or a pair of sequence for sequence classification tasks by concatenating and adding special tokens. A BERT sequence has the following format:

* single sequence: [CLS] X [SEP]
* pair of sequences: [CLS] A [SEP] B [SEP]

get\_special\_tokens\_mask

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/tokenization_bert.py#L295)

( token\_ids\_0: typing.List[int]token\_ids\_1: typing.Optional[typing.List[int]] = Nonealready\_has\_special\_tokens: bool = False ) **→** List[int]

**Parameters**

* **token\_ids\_0** (List[int]) — List of IDs.
* **token\_ids\_1** (List[int], *optional*) — Optional second list of IDs for sequence pairs.
* **already\_has\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not the token list is already formatted with special tokens for the model.

**Returns**

List[int]

A list of integers in the range [0, 1]: 1 for a special token, 0 for a sequence token.

Retrieve sequence ids from a token list that has no special tokens added. This method is called when adding special tokens using the tokenizer prepare\_for\_model method.

create\_token\_type\_ids\_from\_sequences

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/tokenization_bert.py#L323)

( token\_ids\_0: typing.List[int]token\_ids\_1: typing.Optional[typing.List[int]] = None ) **→** List[int]

**Parameters**

* **token\_ids\_0** (List[int]) — List of IDs.
* **token\_ids\_1** (List[int], *optional*) — Optional second list of IDs for sequence pairs.

**Returns**

List[int]

List of [token type IDs](https://huggingface.co/docs/transformers/main/glossary#token-type-ids) according to the given sequence(s).

Create a mask from the two sequences passed to be used in a sequence-pair classification task. A BERT sequence

pair mask has the following format:

Copied

0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1

| first sequence | second sequence |

If token\_ids\_1 is None, this method only returns the first portion of the mask (0s).

save\_vocabulary

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/tokenization_bert.py#L352)

( save\_directory: strfilename\_prefix: typing.Optional[str] = None )

Pytorch

Hide Pytorch content

**BertTokenizerFast**

classtransformers.**BertTokenizerFast**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/tokenization_bert_fast.py#L161)

( vocab\_file = Nonetokenizer\_file = Nonedo\_lower\_case = Trueunk\_token = '[UNK]'sep\_token = '[SEP]'pad\_token = '[PAD]'cls\_token = '[CLS]'mask\_token = '[MASK]'tokenize\_chinese\_chars = Truestrip\_accents = None\*\*kwargs )

Expand 11 parameters

**Parameters**

* **vocab\_file** (str) — File containing the vocabulary.
* **do\_lower\_case** (bool, *optional*, defaults to True) — Whether or not to lowercase the input when tokenizing.
* **unk\_token** (str, *optional*, defaults to "[UNK]") — The unknown token. A token that is not in the vocabulary cannot be converted to an ID and is set to be this token instead.
* **sep\_token** (str, *optional*, defaults to "[SEP]") — The separator token, which is used when building a sequence from multiple sequences, e.g. two sequences for sequence classification or for a text and a question for question answering. It is also used as the last token of a sequence built with special tokens.
* **pad\_token** (str, *optional*, defaults to "[PAD]") — The token used for padding, for example when batching sequences of different lengths.
* **cls\_token** (str, *optional*, defaults to "[CLS]") — The classifier token which is used when doing sequence classification (classification of the whole sequence instead of per-token classification). It is the first token of the sequence when built with special tokens.
* **mask\_token** (str, *optional*, defaults to "[MASK]") — The token used for masking values. This is the token used when training this model with masked language modeling. This is the token which the model will try to predict.
* **clean\_text** (bool, *optional*, defaults to True) — Whether or not to clean the text before tokenization by removing any control characters and replacing all whitespaces by the classic one.
* **tokenize\_chinese\_chars** (bool, *optional*, defaults to True) — Whether or not to tokenize Chinese characters. This should likely be deactivated for Japanese (see [this issue](https://github.com/huggingface/transformers/issues/328)).
* **strip\_accents** (bool, *optional*) — Whether or not to strip all accents. If this option is not specified, then it will be determined by the value for lowercase (as in the original BERT).
* **wordpieces\_prefix** (str, *optional*, defaults to "##") — The prefix for subwords.

Construct a “fast” BERT tokenizer (backed by HuggingFace’s *tokenizers* library). Based on WordPiece.

This tokenizer inherits from [PreTrainedTokenizerFast](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast) which contains most of the main methods. Users should refer to this superclass for more information regarding those methods.

build\_inputs\_with\_special\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/tokenization_bert_fast.py#L249)

( token\_ids\_0token\_ids\_1 = None ) **→** List[int]

**Parameters**

* **token\_ids\_0** (List[int]) — List of IDs to which the special tokens will be added.
* **token\_ids\_1** (List[int], *optional*) — Optional second list of IDs for sequence pairs.

**Returns**

List[int]

List of [input IDs](https://huggingface.co/docs/transformers/main/glossary#input-ids) with the appropriate special tokens.

Build model inputs from a sequence or a pair of sequence for sequence classification tasks by concatenating and adding special tokens. A BERT sequence has the following format:

* single sequence: [CLS] X [SEP]
* pair of sequences: [CLS] A [SEP] B [SEP]

create\_token\_type\_ids\_from\_sequences

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/tokenization_bert_fast.py#L273)

( token\_ids\_0: typing.List[int]token\_ids\_1: typing.Optional[typing.List[int]] = None ) **→** List[int]

**Parameters**

* **token\_ids\_0** (List[int]) — List of IDs.
* **token\_ids\_1** (List[int], *optional*) — Optional second list of IDs for sequence pairs.

**Returns**

List[int]

List of [token type IDs](https://huggingface.co/docs/transformers/main/glossary#token-type-ids) according to the given sequence(s).

Create a mask from the two sequences passed to be used in a sequence-pair classification task. A BERT sequence

pair mask has the following format:

Copied

0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1

| first sequence | second sequence |

If token\_ids\_1 is None, this method only returns the first portion of the mask (0s).

TensorFlow

Hide TensorFlow content

**TFBertTokenizer**

classtransformers.**TFBertTokenizer**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/tokenization_bert_tf.py#L11)

( \*args\*\*kwargs )

Expand 12 parameters

**Parameters**

* **vocab\_list** (list) — List containing the vocabulary.
* **do\_lower\_case** (bool, *optional*, defaults to True) — Whether or not to lowercase the input when tokenizing.
* **cls\_token\_id** (str, *optional*, defaults to "[CLS]") — The classifier token which is used when doing sequence classification (classification of the whole sequence instead of per-token classification). It is the first token of the sequence when built with special tokens.
* **sep\_token\_id** (str, *optional*, defaults to "[SEP]") — The separator token, which is used when building a sequence from multiple sequences, e.g. two sequences for sequence classification or for a text and a question for question answering. It is also used as the last token of a sequence built with special tokens.
* **pad\_token\_id** (str, *optional*, defaults to "[PAD]") — The token used for padding, for example when batching sequences of different lengths.
* **padding** (str, defaults to "longest") — The type of padding to use. Can be either "longest", to pad only up to the longest sample in the batch, or `“max\_length”, to pad all inputs to the maximum length supported by the tokenizer.
* **truncation** (bool, *optional*, defaults to True) — Whether to truncate the sequence to the maximum length.
* **max\_length** (int, *optional*, defaults to 512) — The maximum length of the sequence, used for padding (if padding is “max\_length”) and/or truncation (if truncation is True).
* **pad\_to\_multiple\_of** (int, *optional*, defaults to None) — If set, the sequence will be padded to a multiple of this value.
* **return\_token\_type\_ids** (bool, *optional*, defaults to True) — Whether to return token\_type\_ids.
* **return\_attention\_mask** (bool, *optional*, defaults to True) — Whether to return the attention\_mask.
* **use\_fast\_bert\_tokenizer** (bool, *optional*, defaults to True) — If True, will use the FastBertTokenizer class from Tensorflow Text. If False, will use the BertTokenizer class instead. BertTokenizer supports some additional options, but is slower and cannot be exported to TFLite.

This is an in-graph tokenizer for BERT. It should be initialized similarly to other tokenizers, using the from\_pretrained() method. It can also be initialized with the from\_tokenizer() method, which imports settings from an existing standard tokenizer object.

In-graph tokenizers, unlike other Hugging Face tokenizers, are actually Keras layers and are designed to be run when the model is called, rather than during preprocessing. As a result, they have somewhat more limited options than standard tokenizer classes. They are most useful when you want to create an end-to-end model that goes straight from tf.string inputs to outputs.

from\_pretrained

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/tokenization_bert_tf.py#L143)

( pretrained\_model\_name\_or\_path: typing.Union[str, os.PathLike]\*init\_inputs\*\*kwargs )

**Parameters**

* **pretrained\_model\_name\_or\_path** (str or os.PathLike) — The name or path to the pre-trained tokenizer.

Instantiate a TFBertTokenizer from a pre-trained tokenizer.

Examples:

Copied

from transformers import TFBertTokenizer

tf\_tokenizer = TFBertTokenizer.from\_pretrained("bert-base-uncased")

from\_tokenizer

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/tokenization_bert_tf.py#L104)

( tokenizer: PreTrainedTokenizerBase\*\*kwargs )

**Parameters**

* **tokenizer** (PreTrainedTokenizerBase) — The tokenizer to use to initialize the TFBertTokenizer.

Initialize a TFBertTokenizer from an existing Tokenizer.

Examples:

Copied

from transformers import AutoTokenizer, TFBertTokenizer

tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

tf\_tokenizer = TFBertTokenizer.from\_tokenizer(tokenizer)

**Bert specific outputs**

classtransformers.models.bert.modeling\_bert.**BertForPreTrainingOutput**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L762)

( loss: typing.Optional[torch.FloatTensor] = Noneprediction\_logits: FloatTensor = Noneseq\_relationship\_logits: FloatTensor = Nonehidden\_states: typing.Optional[typing.Tuple[torch.FloatTensor]] = Noneattentions: typing.Optional[typing.Tuple[torch.FloatTensor]] = None )

**Parameters**

* **loss** (*optional*, returned when labels is provided, torch.FloatTensor of shape (1,)) — Total loss as the sum of the masked language modeling loss and the next sequence prediction (classification) loss.
* **prediction\_logits** (torch.FloatTensor of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **seq\_relationship\_logits** (torch.FloatTensor of shape (batch\_size, 2)) — Prediction scores of the next sequence prediction (classification) head (scores of True/False continuation before SoftMax).
* **hidden\_states** (tuple(torch.FloatTensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of torch.FloatTensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

Output type of [BertForPreTraining](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForPreTraining).

classtransformers.models.bert.modeling\_tf\_bert.**TFBertForPreTrainingOutput**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1053)

( loss: tf.Tensor | None = Noneprediction\_logits: tf.Tensor = Noneseq\_relationship\_logits: tf.Tensor = Nonehidden\_states: Optional[Union[Tuple[tf.Tensor], tf.Tensor]] = Noneattentions: Optional[Union[Tuple[tf.Tensor], tf.Tensor]] = None )

**Parameters**

* **prediction\_logits** (tf.Tensor of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **seq\_relationship\_logits** (tf.Tensor of shape (batch\_size, 2)) — Prediction scores of the next sequence prediction (classification) head (scores of True/False continuation before SoftMax).
* **hidden\_states** (tuple(tf.Tensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of tf.Tensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

Output type of [TFBertForPreTraining](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForPreTraining).

classtransformers.models.bert.modeling\_flax\_bert.**FlaxBertForPreTrainingOutput**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L62)

( prediction\_logits: Array = Noneseq\_relationship\_logits: Array = Nonehidden\_states: typing.Optional[typing.Tuple[jax.Array]] = Noneattentions: typing.Optional[typing.Tuple[jax.Array]] = None )

**Parameters**

* **prediction\_logits** (jnp.ndarray of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **seq\_relationship\_logits** (jnp.ndarray of shape (batch\_size, 2)) — Prediction scores of the next sequence prediction (classification) head (scores of True/False continuation before SoftMax).
* **hidden\_states** (tuple(jnp.ndarray), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of jnp.ndarray (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(jnp.ndarray), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of jnp.ndarray (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

Output type of [BertForPreTraining](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForPreTraining).

replace

[<source>](https://github.com/huggingface/transformers/blob/main/src/flax/struct.py#L111)

( \*\*updates )

“Returns a new object replacing the specified fields with new values.

Pytorch

Hide Pytorch content

**BertModel**

classtransformers.**BertModel**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L865)

( configadd\_pooling\_layer = True )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.from_pretrained) method to load the model weights.

The bare Bert Model transformer outputting raw hidden-states without any specific head on top.

This model inherits from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch [torch.nn.Module](https://pytorch.org/docs/stable/nn.html#torch.nn.Module) subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

The model can behave as an encoder (with only self-attention) as well as a decoder, in which case a layer of cross-attention is added between the self-attention layers, following the architecture described in [Attention is all you need](https://arxiv.org/abs/1706.03762) by Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin.

To behave as an decoder the model needs to be initialized with the is\_decoder argument of the configuration set to True. To be used in a Seq2Seq model, the model needs to initialized with both is\_decoder argument and add\_cross\_attention set to True; an encoder\_hidden\_states is then expected as an input to the forward pass.

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L904)

( input\_ids: typing.Optional[torch.Tensor] = Noneattention\_mask: typing.Optional[torch.Tensor] = Nonetoken\_type\_ids: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.Tensor] = Nonehead\_mask: typing.Optional[torch.Tensor] = Noneinputs\_embeds: typing.Optional[torch.Tensor] = Noneencoder\_hidden\_states: typing.Optional[torch.Tensor] = Noneencoder\_attention\_mask: typing.Optional[torch.Tensor] = Nonepast\_key\_values: typing.Optional[typing.List[torch.FloatTensor]] = Noneuse\_cache: typing.Optional[bool] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None ) **→** [transformers.modeling\_outputs.BaseModelOutputWithPoolingAndCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.BaseModelOutputWithPoolingAndCrossAttentions) or tuple(torch.FloatTensor)

Expand 13 parameters

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.FloatTensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (torch.FloatTensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.
* **encoder\_hidden\_states** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Sequence of hidden-states at the output of the last layer of the encoder. Used in the cross-attention if the model is configured as a decoder.
* **encoder\_attention\_mask** (torch.FloatTensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on the padding token indices of the encoder input. This mask is used in the cross-attention if the model is configured as a decoder. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.
* **past\_key\_values** (tuple(tuple(torch.FloatTensor)) of length config.n\_layers with each tuple having 4 tensors of shape (batch\_size, num\_heads, sequence\_length - 1, embed\_size\_per\_head)) — Contains precomputed key and value hidden states of the attention blocks. Can be used to speed up decoding.

If past\_key\_values are used, the user can optionally input only the last decoder\_input\_ids (those that don’t have their past key value states given to this model) of shape (batch\_size, 1) instead of all decoder\_input\_ids of shape (batch\_size, sequence\_length).

* **use\_cache** (bool, *optional*) — If set to True, past\_key\_values key value states are returned and can be used to speed up decoding (see past\_key\_values).

**Returns**

[transformers.modeling\_outputs.BaseModelOutputWithPoolingAndCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.BaseModelOutputWithPoolingAndCrossAttentions)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_outputs.BaseModelOutputWithPoolingAndCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.BaseModelOutputWithPoolingAndCrossAttentions) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **last\_hidden\_state** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size)) — Sequence of hidden-states at the output of the last layer of the model.
* **pooler\_output** (torch.FloatTensor of shape (batch\_size, hidden\_size)) — Last layer hidden-state of the first token of the sequence (classification token) after further processing through the layers used for the auxiliary pretraining task. E.g. for BERT-family of models, this returns the classification token after processing through a linear layer and a tanh activation function. The linear layer weights are trained from the next sentence prediction (classification) objective during pretraining.
* **hidden\_states** (tuple(torch.FloatTensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of torch.FloatTensor (one for the output of the embeddings, if the model has an embedding layer, + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the optional initial embedding outputs.

* **attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

* **cross\_attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True and config.add\_cross\_attention=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights of the decoder’s cross-attention layer, after the attention softmax, used to compute the weighted average in the cross-attention heads.

* **past\_key\_values** (tuple(tuple(torch.FloatTensor)), *optional*, returned when use\_cache=True is passed or when config.use\_cache=True) — Tuple of tuple(torch.FloatTensor) of length config.n\_layers, with each tuple having 2 tensors of shape (batch\_size, num\_heads, sequence\_length, embed\_size\_per\_head)) and optionally if config.is\_encoder\_decoder=True 2 additional tensors of shape (batch\_size, num\_heads, encoder\_sequence\_length, embed\_size\_per\_head).

Contains pre-computed hidden-states (key and values in the self-attention blocks and optionally if config.is\_encoder\_decoder=True in the cross-attention blocks) that can be used (see past\_key\_values input) to speed up sequential decoding.

The [BertModel](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertModel) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, BertModel

>>> import torch

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = BertModel.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="pt")

>>> outputs = model(\*\*inputs)

>>> last\_hidden\_states = outputs.last\_hidden\_state

**BertForPreTraining**

classtransformers.**BertForPreTraining**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1048)

( config )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with two heads on top as done during the pretraining: a masked language modeling head and a next sentence prediction (classification) head.

This model inherits from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch [torch.nn.Module](https://pytorch.org/docs/stable/nn.html#torch.nn.Module) subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1066)

( input\_ids: typing.Optional[torch.Tensor] = Noneattention\_mask: typing.Optional[torch.Tensor] = Nonetoken\_type\_ids: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.Tensor] = Nonehead\_mask: typing.Optional[torch.Tensor] = Noneinputs\_embeds: typing.Optional[torch.Tensor] = Nonelabels: typing.Optional[torch.Tensor] = Nonenext\_sentence\_label: typing.Optional[torch.Tensor] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None ) **→** [transformers.models.bert.modeling\_bert.BertForPreTrainingOutput](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.models.bert.modeling_bert.BertForPreTrainingOutput) or tuple(torch.FloatTensor)

Expand 9 parameters

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.FloatTensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (torch.FloatTensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.

labels (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*): Labels for computing the masked language modeling loss. Indices should be in [-100, 0, ..., config.vocab\_size] (see input\_ids docstring) Tokens with indices set to -100 are ignored (masked), the loss is only computed for the tokens with labels in [0, ..., config.vocab\_size] next\_sentence\_label (torch.LongTensor of shape (batch\_size,), *optional*): Labels for computing the next sequence prediction (classification) loss. Input should be a sequence pair (see input\_ids docstring) Indices should be in [0, 1]:

* + 0 indicates sequence B is a continuation of sequence A,
  + 1 indicates sequence B is a random sequence. kwargs (Dict[str, any], optional, defaults to *{}*): Used to hide legacy arguments that have been deprecated.

**Returns**

[transformers.models.bert.modeling\_bert.BertForPreTrainingOutput](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.models.bert.modeling_bert.BertForPreTrainingOutput)**or**tuple(torch.FloatTensor)

A [transformers.models.bert.modeling\_bert.BertForPreTrainingOutput](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.models.bert.modeling_bert.BertForPreTrainingOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (*optional*, returned when labels is provided, torch.FloatTensor of shape (1,)) — Total loss as the sum of the masked language modeling loss and the next sequence prediction (classification) loss.
* **prediction\_logits** (torch.FloatTensor of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **seq\_relationship\_logits** (torch.FloatTensor of shape (batch\_size, 2)) — Prediction scores of the next sequence prediction (classification) head (scores of True/False continuation before SoftMax).
* **hidden\_states** (tuple(torch.FloatTensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of torch.FloatTensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [BertForPreTraining](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForPreTraining) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, BertForPreTraining

>>> import torch

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = BertForPreTraining.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="pt")

>>> outputs = model(\*\*inputs)

>>> prediction\_logits = outputs.prediction\_logits

>>> seq\_relationship\_logits = outputs.seq\_relationship\_logits

**BertLMHeadModel**

classtransformers.**BertLMHeadModel**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1154)

( config )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with a language modeling head on top for CLM fine-tuning.

This model inherits from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch [torch.nn.Module](https://pytorch.org/docs/stable/nn.html#torch.nn.Module) subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1175)

( input\_ids: typing.Optional[torch.Tensor] = Noneattention\_mask: typing.Optional[torch.Tensor] = Nonetoken\_type\_ids: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.Tensor] = Nonehead\_mask: typing.Optional[torch.Tensor] = Noneinputs\_embeds: typing.Optional[torch.Tensor] = Noneencoder\_hidden\_states: typing.Optional[torch.Tensor] = Noneencoder\_attention\_mask: typing.Optional[torch.Tensor] = Nonelabels: typing.Optional[torch.Tensor] = Nonepast\_key\_values: typing.Optional[typing.List[torch.Tensor]] = Noneuse\_cache: typing.Optional[bool] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None ) **→** [transformers.modeling\_outputs.CausalLMOutputWithCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.CausalLMOutputWithCrossAttentions) or tuple(torch.FloatTensor)

Expand 14 parameters

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.FloatTensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (torch.FloatTensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.
* **encoder\_hidden\_states** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Sequence of hidden-states at the output of the last layer of the encoder. Used in the cross-attention if the model is configured as a decoder.
* **encoder\_attention\_mask** (torch.FloatTensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on the padding token indices of the encoder input. This mask is used in the cross-attention if the model is configured as a decoder. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.
* **labels** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Labels for computing the left-to-right language modeling loss (next word prediction). Indices should be in [-100, 0, ..., config.vocab\_size] (see input\_ids docstring) Tokens with indices set to -100 are ignored (masked), the loss is only computed for the tokens with labels n [0, ..., config.vocab\_size]
* **past\_key\_values** (tuple(tuple(torch.FloatTensor)) of length config.n\_layers with each tuple having 4 tensors of shape (batch\_size, num\_heads, sequence\_length - 1, embed\_size\_per\_head)) — Contains precomputed key and value hidden states of the attention blocks. Can be used to speed up decoding.

If past\_key\_values are used, the user can optionally input only the last decoder\_input\_ids (those that don’t have their past key value states given to this model) of shape (batch\_size, 1) instead of all decoder\_input\_ids of shape (batch\_size, sequence\_length).

* **use\_cache** (bool, *optional*) — If set to True, past\_key\_values key value states are returned and can be used to speed up decoding (see past\_key\_values).

**Returns**

[transformers.modeling\_outputs.CausalLMOutputWithCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.CausalLMOutputWithCrossAttentions)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_outputs.CausalLMOutputWithCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.CausalLMOutputWithCrossAttentions) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (torch.FloatTensor of shape (1,), *optional*, returned when labels is provided) — Language modeling loss (for next-token prediction).
* **logits** (torch.FloatTensor of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **hidden\_states** (tuple(torch.FloatTensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of torch.FloatTensor (one for the output of the embeddings, if the model has an embedding layer, + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the optional initial embedding outputs.

* **attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

* **cross\_attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Cross attentions weights after the attention softmax, used to compute the weighted average in the cross-attention heads.

* **past\_key\_values** (tuple(tuple(torch.FloatTensor)), *optional*, returned when use\_cache=True is passed or when config.use\_cache=True) — Tuple of torch.FloatTensor tuples of length config.n\_layers, with each tuple containing the cached key, value states of the self-attention and the cross-attention layers if model is used in encoder-decoder setting. Only relevant if config.is\_decoder = True.

Contains pre-computed hidden-states (key and values in the attention blocks) that can be used (see past\_key\_values input) to speed up sequential decoding.

The [BertLMHeadModel](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertLMHeadModel) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> import torch

>>> from transformers import AutoTokenizer, BertLMHeadModel

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = BertLMHeadModel.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="pt")

>>> outputs = model(\*\*inputs, labels=inputs["input\_ids"])

>>> loss = outputs.loss

>>> logits = outputs.logits

**BertForMaskedLM**

classtransformers.**BertForMaskedLM**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1304)

( config )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with a language modeling head on top.

This model inherits from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch [torch.nn.Module](https://pytorch.org/docs/stable/nn.html#torch.nn.Module) subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1328)

( input\_ids: typing.Optional[torch.Tensor] = Noneattention\_mask: typing.Optional[torch.Tensor] = Nonetoken\_type\_ids: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.Tensor] = Nonehead\_mask: typing.Optional[torch.Tensor] = Noneinputs\_embeds: typing.Optional[torch.Tensor] = Noneencoder\_hidden\_states: typing.Optional[torch.Tensor] = Noneencoder\_attention\_mask: typing.Optional[torch.Tensor] = Nonelabels: typing.Optional[torch.Tensor] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None ) **→** [transformers.modeling\_outputs.MaskedLMOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.MaskedLMOutput) or tuple(torch.FloatTensor)

Expand 10 parameters

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.FloatTensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (torch.FloatTensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.
* **labels** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Labels for computing the masked language modeling loss. Indices should be in [-100, 0, ..., config.vocab\_size] (see input\_ids docstring) Tokens with indices set to -100 are ignored (masked), the loss is only computed for the tokens with labels in [0, ..., config.vocab\_size]

**Returns**

[transformers.modeling\_outputs.MaskedLMOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.MaskedLMOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_outputs.MaskedLMOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.MaskedLMOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (torch.FloatTensor of shape (1,), *optional*, returned when labels is provided) — Masked language modeling (MLM) loss.
* **logits** (torch.FloatTensor of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **hidden\_states** (tuple(torch.FloatTensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of torch.FloatTensor (one for the output of the embeddings, if the model has an embedding layer, + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the optional initial embedding outputs.

* **attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [BertForMaskedLM](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForMaskedLM) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, BertForMaskedLM

>>> import torch

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = BertForMaskedLM.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("The capital of France is [MASK].", return\_tensors="pt")

>>> with torch.no\_grad():

... logits = model(\*\*inputs).logits

>>> *# retrieve index of [MASK]*

>>> mask\_token\_index = (inputs.input\_ids == tokenizer.mask\_token\_id)[0].nonzero(as\_tuple=True)[0]

>>> predicted\_token\_id = logits[0, mask\_token\_index].argmax(axis=-1)

>>> tokenizer.decode(predicted\_token\_id)

'paris'

>>> labels = tokenizer("The capital of France is Paris.", return\_tensors="pt")["input\_ids"]

>>> *# mask labels of non-[MASK] tokens*

>>> labels = torch.where(inputs.input\_ids == tokenizer.mask\_token\_id, labels, -100)

>>> outputs = model(\*\*inputs, labels=labels)

>>> round(outputs.loss.item(), 2)

0.88

**BertForNextSentencePrediction**

classtransformers.**BertForNextSentencePrediction**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1414)

( config )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with a next sentence prediction (classification) head on top.

This model inherits from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch [torch.nn.Module](https://pytorch.org/docs/stable/nn.html#torch.nn.Module) subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1424)

( input\_ids: typing.Optional[torch.Tensor] = Noneattention\_mask: typing.Optional[torch.Tensor] = Nonetoken\_type\_ids: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.Tensor] = Nonehead\_mask: typing.Optional[torch.Tensor] = Noneinputs\_embeds: typing.Optional[torch.Tensor] = Nonelabels: typing.Optional[torch.Tensor] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None\*\*kwargs ) **→** [transformers.modeling\_outputs.NextSentencePredictorOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.NextSentencePredictorOutput) or tuple(torch.FloatTensor)

Expand 10 parameters

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.FloatTensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (torch.FloatTensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.
* **labels** (torch.LongTensor of shape (batch\_size,), *optional*) — Labels for computing the next sequence prediction (classification) loss. Input should be a sequence pair (see input\_ids docstring). Indices should be in [0, 1]:
  + 0 indicates sequence B is a continuation of sequence A,
  + 1 indicates sequence B is a random sequence.

**Returns**

[transformers.modeling\_outputs.NextSentencePredictorOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.NextSentencePredictorOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_outputs.NextSentencePredictorOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.NextSentencePredictorOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (torch.FloatTensor of shape (1,), *optional*, returned when next\_sentence\_label is provided) — Next sequence prediction (classification) loss.
* **logits** (torch.FloatTensor of shape (batch\_size, 2)) — Prediction scores of the next sequence prediction (classification) head (scores of True/False continuation before SoftMax).
* **hidden\_states** (tuple(torch.FloatTensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of torch.FloatTensor (one for the output of the embeddings, if the model has an embedding layer, + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the optional initial embedding outputs.

* **attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [BertForNextSentencePrediction](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForNextSentencePrediction) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, BertForNextSentencePrediction

>>> import torch

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = BertForNextSentencePrediction.from\_pretrained("bert-base-uncased")

>>> prompt = "In Italy, pizza served in formal settings, such as at a restaurant, is presented unsliced."

>>> next\_sentence = "The sky is blue due to the shorter wavelength of blue light."

>>> encoding = tokenizer(prompt, next\_sentence, return\_tensors="pt")

>>> outputs = model(\*\*encoding, labels=torch.LongTensor([1]))

>>> logits = outputs.logits

>>> assert logits[0, 0] < logits[0, 1] *# next sentence was random*

**BertForSequenceClassification**

classtransformers.**BertForSequenceClassification**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1519)

( config )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.from_pretrained) method to load the model weights.

Bert Model transformer with a sequence classification/regression head on top (a linear layer on top of the pooled output) e.g. for GLUE tasks.

This model inherits from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch [torch.nn.Module](https://pytorch.org/docs/stable/nn.html#torch.nn.Module) subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1535)

( input\_ids: typing.Optional[torch.Tensor] = Noneattention\_mask: typing.Optional[torch.Tensor] = Nonetoken\_type\_ids: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.Tensor] = Nonehead\_mask: typing.Optional[torch.Tensor] = Noneinputs\_embeds: typing.Optional[torch.Tensor] = Nonelabels: typing.Optional[torch.Tensor] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None ) **→** [transformers.modeling\_outputs.SequenceClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.SequenceClassifierOutput) or tuple(torch.FloatTensor)

Expand 10 parameters

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.FloatTensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (torch.FloatTensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.
* **labels** (torch.LongTensor of shape (batch\_size,), *optional*) — Labels for computing the sequence classification/regression loss. Indices should be in [0, ..., config.num\_labels - 1]. If config.num\_labels == 1 a regression loss is computed (Mean-Square loss), If config.num\_labels > 1 a classification loss is computed (Cross-Entropy).

**Returns**

[transformers.modeling\_outputs.SequenceClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.SequenceClassifierOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_outputs.SequenceClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.SequenceClassifierOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (torch.FloatTensor of shape (1,), *optional*, returned when labels is provided) — Classification (or regression if config.num\_labels==1) loss.
* **logits** (torch.FloatTensor of shape (batch\_size, config.num\_labels)) — Classification (or regression if config.num\_labels==1) scores (before SoftMax).
* **hidden\_states** (tuple(torch.FloatTensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of torch.FloatTensor (one for the output of the embeddings, if the model has an embedding layer, + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the optional initial embedding outputs.

* **attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [BertForSequenceClassification](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForSequenceClassification) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example of single-label classification:

Copied

>>> import torch

>>> from transformers import AutoTokenizer, BertForSequenceClassification

>>> tokenizer = AutoTokenizer.from\_pretrained("textattack/bert-base-uncased-yelp-polarity")

>>> model = BertForSequenceClassification.from\_pretrained("textattack/bert-base-uncased-yelp-polarity")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="pt")

>>> with torch.no\_grad():

... logits = model(\*\*inputs).logits

>>> predicted\_class\_id = logits.argmax().item()

>>> model.config.id2label[predicted\_class\_id]

'LABEL\_1'

>>> *# To train a model on `num\_labels` classes, you can pass `num\_labels=num\_labels` to `.from\_pretrained(...)`*

>>> num\_labels = len(model.config.id2label)

>>> model = BertForSequenceClassification.from\_pretrained("textattack/bert-base-uncased-yelp-polarity", num\_labels=num\_labels)

>>> labels = torch.tensor([1])

>>> loss = model(\*\*inputs, labels=labels).loss

>>> round(loss.item(), 2)

0.01

Example of multi-label classification:

Copied

>>> import torch

>>> from transformers import AutoTokenizer, BertForSequenceClassification

>>> tokenizer = AutoTokenizer.from\_pretrained("textattack/bert-base-uncased-yelp-polarity")

>>> model = BertForSequenceClassification.from\_pretrained("textattack/bert-base-uncased-yelp-polarity", problem\_type="multi\_label\_classification")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="pt")

>>> with torch.no\_grad():

... logits = model(\*\*inputs).logits

>>> predicted\_class\_ids = torch.arange(0, logits.shape[-1])[torch.sigmoid(logits).squeeze(dim=0) > 0.5]

>>> *# To train a model on `num\_labels` classes, you can pass `num\_labels=num\_labels` to `.from\_pretrained(...)`*

>>> num\_labels = len(model.config.id2label)

>>> model = BertForSequenceClassification.from\_pretrained(

... "textattack/bert-base-uncased-yelp-polarity", num\_labels=num\_labels, problem\_type="multi\_label\_classification"

... )

>>> labels = torch.sum(

... torch.nn.functional.one\_hot(predicted\_class\_ids[None, :].clone(), num\_classes=num\_labels), dim=1

... ).to(torch.float)

>>> loss = model(\*\*inputs, labels=labels).loss

**BertForMultipleChoice**

classtransformers.**BertForMultipleChoice**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1622)

( config )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with a multiple choice classification head on top (a linear layer on top of the pooled output and a softmax) e.g. for RocStories/SWAG tasks.

This model inherits from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch [torch.nn.Module](https://pytorch.org/docs/stable/nn.html#torch.nn.Module) subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1636)

( input\_ids: typing.Optional[torch.Tensor] = Noneattention\_mask: typing.Optional[torch.Tensor] = Nonetoken\_type\_ids: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.Tensor] = Nonehead\_mask: typing.Optional[torch.Tensor] = Noneinputs\_embeds: typing.Optional[torch.Tensor] = Nonelabels: typing.Optional[torch.Tensor] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None ) **→** [transformers.modeling\_outputs.MultipleChoiceModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.MultipleChoiceModelOutput) or tuple(torch.FloatTensor)

Expand 10 parameters

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, num\_choices, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.FloatTensor of shape (batch\_size, num\_choices, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (torch.LongTensor of shape (batch\_size, num\_choices, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (torch.LongTensor of shape (batch\_size, num\_choices, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (torch.FloatTensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, num\_choices, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.
* **labels** (torch.LongTensor of shape (batch\_size,), *optional*) — Labels for computing the multiple choice classification loss. Indices should be in [0, ..., num\_choices-1] where num\_choices is the size of the second dimension of the input tensors. (See input\_ids above)

**Returns**

[transformers.modeling\_outputs.MultipleChoiceModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.MultipleChoiceModelOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_outputs.MultipleChoiceModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.MultipleChoiceModelOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (torch.FloatTensor of shape *(1,)*, *optional*, returned when labels is provided) — Classification loss.
* **logits** (torch.FloatTensor of shape (batch\_size, num\_choices)) — *num\_choices* is the second dimension of the input tensors. (see *input\_ids* above).

Classification scores (before SoftMax).

* **hidden\_states** (tuple(torch.FloatTensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of torch.FloatTensor (one for the output of the embeddings, if the model has an embedding layer, + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the optional initial embedding outputs.

* **attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [BertForMultipleChoice](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForMultipleChoice) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, BertForMultipleChoice

>>> import torch

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = BertForMultipleChoice.from\_pretrained("bert-base-uncased")

>>> prompt = "In Italy, pizza served in formal settings, such as at a restaurant, is presented unsliced."

>>> choice0 = "It is eaten with a fork and a knife."

>>> choice1 = "It is eaten while held in the hand."

>>> labels = torch.tensor(0).unsqueeze(0) *# choice0 is correct (according to Wikipedia ;)), batch size 1*

>>> encoding = tokenizer([prompt, prompt], [choice0, choice1], return\_tensors="pt", padding=True)

>>> outputs = model(\*\*{k: v.unsqueeze(0) for k, v in encoding.items()}, labels=labels) *# batch size is 1*

>>> *# the linear classifier still needs to be trained*

>>> loss = outputs.loss

>>> logits = outputs.logits

**BertForTokenClassification**

classtransformers.**BertForTokenClassification**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1716)

( config )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with a token classification head on top (a linear layer on top of the hidden-states output) e.g. for Named-Entity-Recognition (NER) tasks.

This model inherits from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch [torch.nn.Module](https://pytorch.org/docs/stable/nn.html#torch.nn.Module) subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1731)

( input\_ids: typing.Optional[torch.Tensor] = Noneattention\_mask: typing.Optional[torch.Tensor] = Nonetoken\_type\_ids: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.Tensor] = Nonehead\_mask: typing.Optional[torch.Tensor] = Noneinputs\_embeds: typing.Optional[torch.Tensor] = Nonelabels: typing.Optional[torch.Tensor] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None ) **→** [transformers.modeling\_outputs.TokenClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.TokenClassifierOutput) or tuple(torch.FloatTensor)

Expand 10 parameters

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.FloatTensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (torch.FloatTensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.
* **labels** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Labels for computing the token classification loss. Indices should be in [0, ..., config.num\_labels - 1].

**Returns**

[transformers.modeling\_outputs.TokenClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.TokenClassifierOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_outputs.TokenClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.TokenClassifierOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (torch.FloatTensor of shape (1,), *optional*, returned when labels is provided) — Classification loss.
* **logits** (torch.FloatTensor of shape (batch\_size, sequence\_length, config.num\_labels)) — Classification scores (before SoftMax).
* **hidden\_states** (tuple(torch.FloatTensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of torch.FloatTensor (one for the output of the embeddings, if the model has an embedding layer, + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the optional initial embedding outputs.

* **attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [BertForTokenClassification](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForTokenClassification) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, BertForTokenClassification

>>> import torch

>>> tokenizer = AutoTokenizer.from\_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")

>>> model = BertForTokenClassification.from\_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")

>>> inputs = tokenizer(

... "HuggingFace is a company based in Paris and New York", add\_special\_tokens=False, return\_tensors="pt"

... )

>>> with torch.no\_grad():

... logits = model(\*\*inputs).logits

>>> predicted\_token\_class\_ids = logits.argmax(-1)

>>> *# Note that tokens are classified rather then input words which means that*

>>> *# there might be more predicted token classes than words.*

>>> *# Multiple token classes might account for the same word*

>>> predicted\_tokens\_classes = [model.config.id2label[t.item()] for t in predicted\_token\_class\_ids[0]]

>>> predicted\_tokens\_classes

['O', 'I-ORG', 'I-ORG', 'I-ORG', 'O', 'O', 'O', 'O', 'O', 'I-LOC', 'O', 'I-LOC', 'I-LOC']

>>> labels = predicted\_token\_class\_ids

>>> loss = model(\*\*inputs, labels=labels).loss

>>> round(loss.item(), 2)

0.01

**BertForQuestionAnswering**

classtransformers.**BertForQuestionAnswering**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1799)

( config )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with a span classification head on top for extractive question-answering tasks like SQuAD (a linear layers on top of the hidden-states output to compute span start logits and span end logits).

This model inherits from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch [torch.nn.Module](https://pytorch.org/docs/stable/nn.html#torch.nn.Module) subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1810)

( input\_ids: typing.Optional[torch.Tensor] = Noneattention\_mask: typing.Optional[torch.Tensor] = Nonetoken\_type\_ids: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.Tensor] = Nonehead\_mask: typing.Optional[torch.Tensor] = Noneinputs\_embeds: typing.Optional[torch.Tensor] = Nonestart\_positions: typing.Optional[torch.Tensor] = Noneend\_positions: typing.Optional[torch.Tensor] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None ) **→** [transformers.modeling\_outputs.QuestionAnsweringModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.QuestionAnsweringModelOutput) or tuple(torch.FloatTensor)

Expand 11 parameters

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.FloatTensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (torch.FloatTensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.
* **start\_positions** (torch.LongTensor of shape (batch\_size,), *optional*) — Labels for position (index) of the start of the labelled span for computing the token classification loss. Positions are clamped to the length of the sequence (sequence\_length). Position outside of the sequence are not taken into account for computing the loss.
* **end\_positions** (torch.LongTensor of shape (batch\_size,), *optional*) — Labels for position (index) of the end of the labelled span for computing the token classification loss. Positions are clamped to the length of the sequence (sequence\_length). Position outside of the sequence are not taken into account for computing the loss.

**Returns**

[transformers.modeling\_outputs.QuestionAnsweringModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.QuestionAnsweringModelOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_outputs.QuestionAnsweringModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.QuestionAnsweringModelOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (torch.FloatTensor of shape (1,), *optional*, returned when labels is provided) — Total span extraction loss is the sum of a Cross-Entropy for the start and end positions.
* **start\_logits** (torch.FloatTensor of shape (batch\_size, sequence\_length)) — Span-start scores (before SoftMax).
* **end\_logits** (torch.FloatTensor of shape (batch\_size, sequence\_length)) — Span-end scores (before SoftMax).
* **hidden\_states** (tuple(torch.FloatTensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of torch.FloatTensor (one for the output of the embeddings, if the model has an embedding layer, + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the optional initial embedding outputs.

* **attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [BertForQuestionAnswering](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertForQuestionAnswering) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, BertForQuestionAnswering

>>> import torch

>>> tokenizer = AutoTokenizer.from\_pretrained("deepset/bert-base-cased-squad2")

>>> model = BertForQuestionAnswering.from\_pretrained("deepset/bert-base-cased-squad2")

>>> question, text = "Who was Jim Henson?", "Jim Henson was a nice puppet"

>>> inputs = tokenizer(question, text, return\_tensors="pt")

>>> with torch.no\_grad():

... outputs = model(\*\*inputs)

>>> answer\_start\_index = outputs.start\_logits.argmax()

>>> answer\_end\_index = outputs.end\_logits.argmax()

>>> predict\_answer\_tokens = inputs.input\_ids[0, answer\_start\_index : answer\_end\_index + 1]

>>> tokenizer.decode(predict\_answer\_tokens, skip\_special\_tokens=True)

'a nice puppet'

>>> *# target is "nice puppet"*

>>> target\_start\_index = torch.tensor([14])

>>> target\_end\_index = torch.tensor([15])

>>> outputs = model(\*\*inputs, start\_positions=target\_start\_index, end\_positions=target\_end\_index)

>>> loss = outputs.loss

>>> round(loss.item(), 2)

7.41

TensorFlow

Hide TensorFlow content

**TFBertModel**

classtransformers.**TFBertModel**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1185)

( \*args\*\*kwargs )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel.from_pretrained) method to load the model weights.

The bare Bert Model transformer outputting raw hidden-states without any specific head on top.

This model inherits from [TFPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a [tf.keras.Model](https://www.tensorflow.org/api_docs/python/tf/keras/Model) subclass. Use it as a regular TF 2.0 Keras Model and refer to the TF 2.0 documentation for all matter related to general usage and behavior.

TensorFlow models and layers in transformers accept two formats as input:

* having all inputs as keyword arguments (like PyTorch models), or
* having all inputs as a list, tuple or dict in the first positional argument.

The reason the second format is supported is that Keras methods prefer this format when passing inputs to models and layers. Because of this support, when using methods like model.fit() things should “just work” for you - just pass your inputs and labels in any format that model.fit() supports! If, however, you want to use the second format outside of Keras methods like fit() and predict(), such as when creating your own layers or models with the Keras Functional API, there are three possibilities you can use to gather all the input Tensors in the first positional argument:

* a single Tensor with input\_ids only and nothing else: model(input\_ids)
* a list of varying length with one or several input Tensors IN THE ORDER given in the docstring: model([input\_ids, attention\_mask]) or model([input\_ids, attention\_mask, token\_type\_ids])
* a dictionary with one or several input Tensors associated to the input names given in the docstring: model({"input\_ids": input\_ids, "token\_type\_ids": token\_type\_ids})

Note that when creating models and layers with [subclassing](https://keras.io/guides/making_new_layers_and_models_via_subclassing/) then you don’t need to worry about any of this, as you can just pass inputs like you would to any other Python function!

call

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1191)

( input\_ids: TFModelInputType | None = Noneattention\_mask: np.ndarray | tf.Tensor | None = Nonetoken\_type\_ids: np.ndarray | tf.Tensor | None = Noneposition\_ids: np.ndarray | tf.Tensor | None = Nonehead\_mask: np.ndarray | tf.Tensor | None = Noneinputs\_embeds: np.ndarray | tf.Tensor | None = Noneencoder\_hidden\_states: np.ndarray | tf.Tensor | None = Noneencoder\_attention\_mask: np.ndarray | tf.Tensor | None = Nonepast\_key\_values: Optional[Tuple[Tuple[Union[np.ndarray, tf.Tensor]]]] = Noneuse\_cache: Optional[bool] = Noneoutput\_attentions: Optional[bool] = Noneoutput\_hidden\_states: Optional[bool] = Nonereturn\_dict: Optional[bool] = Nonetraining: Optional[bool] = False ) **→** [transformers.modeling\_tf\_outputs.TFBaseModelOutputWithPoolingAndCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFBaseModelOutputWithPoolingAndCrossAttentions) or tuple(tf.Tensor)

Expand 14 parameters

**Parameters**

* **input\_ids** (np.ndarray, tf.Tensor, List[tf.Tensor] `Dict[str, tf.Tensor] or Dict[str, np.ndarray] and each example must have the shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) and [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (np.ndarray or tf.Tensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple. This argument can be used in eager mode, in graph mode the value will always be set to True.
* **training** (bool, *optional*, defaults to `False“) — Whether or not to use the model in training mode (some modules like dropout modules have different behaviors between training and evaluation).
* **encoder\_hidden\_states** (tf.Tensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Sequence of hidden-states at the output of the last layer of the encoder. Used in the cross-attention if the model is configured as a decoder.
* **encoder\_attention\_mask** (tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on the padding token indices of the encoder input. This mask is used in the cross-attention if the model is configured as a decoder. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.
* **past\_key\_values** (Tuple[Tuple[tf.Tensor]] of length config.n\_layers) — contains precomputed key and value hidden states of the attention blocks. Can be used to speed up decoding. If past\_key\_values are used, the user can optionally input only the last decoder\_input\_ids (those that don’t have their past key value states given to this model) of shape (batch\_size, 1) instead of all decoder\_input\_ids of shape (batch\_size, sequence\_length).
* **use\_cache** (bool, *optional*, defaults to True) — If set to True, past\_key\_values key value states are returned and can be used to speed up decoding (see past\_key\_values). Set to False during training, True during generation

**Returns**

[transformers.modeling\_tf\_outputs.TFBaseModelOutputWithPoolingAndCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFBaseModelOutputWithPoolingAndCrossAttentions)**or**tuple(tf.Tensor)

A [transformers.modeling\_tf\_outputs.TFBaseModelOutputWithPoolingAndCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFBaseModelOutputWithPoolingAndCrossAttentions) or a tuple of tf.Tensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **last\_hidden\_state** (tf.Tensor of shape (batch\_size, sequence\_length, hidden\_size)) — Sequence of hidden-states at the output of the last layer of the model.
* **pooler\_output** (tf.Tensor of shape (batch\_size, hidden\_size)) — Last layer hidden-state of the first token of the sequence (classification token) further processed by a Linear layer and a Tanh activation function. The Linear layer weights are trained from the next sentence prediction (classification) objective during pretraining.

This output is usually *not* a good summary of the semantic content of the input, you’re often better with averaging or pooling the sequence of hidden-states for the whole input sequence.

* **past\_key\_values** (List[tf.Tensor], *optional*, returned when use\_cache=True is passed or when config.use\_cache=True) — List of tf.Tensor of length config.n\_layers, with each tensor of shape (2, batch\_size, num\_heads, sequence\_length, embed\_size\_per\_head)).

Contains pre-computed hidden-states (key and values in the attention blocks) that can be used (see past\_key\_values input) to speed up sequential decoding.

* **hidden\_states** (tuple(tf.Tensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of tf.Tensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

* **cross\_attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights of the decoder’s cross-attention layer, after the attention softmax, used to compute the weighted average in the cross-attention heads.

The [TFBertModel](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertModel) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, TFBertModel

>>> import tensorflow as tf

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = TFBertModel.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="tf")

>>> outputs = model(inputs)

>>> last\_hidden\_states = outputs.last\_hidden\_state

**TFBertForPreTraining**

classtransformers.**TFBertForPreTraining**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1269)

( \*args\*\*kwargs )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with two heads on top as done during the pretraining: a masked language modeling head and a next sentence prediction (classification) head.

This model inherits from [TFPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a [tf.keras.Model](https://www.tensorflow.org/api_docs/python/tf/keras/Model) subclass. Use it as a regular TF 2.0 Keras Model and refer to the TF 2.0 documentation for all matter related to general usage and behavior.

TensorFlow models and layers in transformers accept two formats as input:

* having all inputs as keyword arguments (like PyTorch models), or
* having all inputs as a list, tuple or dict in the first positional argument.

The reason the second format is supported is that Keras methods prefer this format when passing inputs to models and layers. Because of this support, when using methods like model.fit() things should “just work” for you - just pass your inputs and labels in any format that model.fit() supports! If, however, you want to use the second format outside of Keras methods like fit() and predict(), such as when creating your own layers or models with the Keras Functional API, there are three possibilities you can use to gather all the input Tensors in the first positional argument:

* a single Tensor with input\_ids only and nothing else: model(input\_ids)
* a list of varying length with one or several input Tensors IN THE ORDER given in the docstring: model([input\_ids, attention\_mask]) or model([input\_ids, attention\_mask, token\_type\_ids])
* a dictionary with one or several input Tensors associated to the input names given in the docstring: model({"input\_ids": input\_ids, "token\_type\_ids": token\_type\_ids})

Note that when creating models and layers with [subclassing](https://keras.io/guides/making_new_layers_and_models_via_subclassing/) then you don’t need to worry about any of this, as you can just pass inputs like you would to any other Python function!

call

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1291)

( input\_ids: TFModelInputType | None = Noneattention\_mask: np.ndarray | tf.Tensor | None = Nonetoken\_type\_ids: np.ndarray | tf.Tensor | None = Noneposition\_ids: np.ndarray | tf.Tensor | None = Nonehead\_mask: np.ndarray | tf.Tensor | None = Noneinputs\_embeds: np.ndarray | tf.Tensor | None = Noneoutput\_attentions: Optional[bool] = Noneoutput\_hidden\_states: Optional[bool] = Nonereturn\_dict: Optional[bool] = Nonelabels: np.ndarray | tf.Tensor | None = Nonenext\_sentence\_label: np.ndarray | tf.Tensor | None = Nonetraining: Optional[bool] = False ) **→** [transformers.models.bert.modeling\_tf\_bert.TFBertForPreTrainingOutput](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.models.bert.modeling_tf_bert.TFBertForPreTrainingOutput) or tuple(tf.Tensor)

Expand 13 parameters

**Parameters**

* **input\_ids** (np.ndarray, tf.Tensor, List[tf.Tensor] `Dict[str, tf.Tensor] or Dict[str, np.ndarray] and each example must have the shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) and [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (np.ndarray or tf.Tensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple. This argument can be used in eager mode, in graph mode the value will always be set to True.
* **training** (bool, *optional*, defaults to `False“) — Whether or not to use the model in training mode (some modules like dropout modules have different behaviors between training and evaluation).
* **labels** (tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Labels for computing the masked language modeling loss. Indices should be in [-100, 0, ..., config.vocab\_size] (see input\_ids docstring) Tokens with indices set to -100 are ignored (masked), the loss is only computed for the tokens with labels in [0, ..., config.vocab\_size]
* **next\_sentence\_label** (tf.Tensor of shape (batch\_size,), *optional*) — Labels for computing the next sequence prediction (classification) loss. Input should be a sequence pair (see input\_ids docstring) Indices should be in [0, 1]:
  + 0 indicates sequence B is a continuation of sequence A,
  + 1 indicates sequence B is a random sequence.
* **kwargs** (Dict[str, any], optional, defaults to *{}*) — Used to hide legacy arguments that have been deprecated.

**Returns**

[transformers.models.bert.modeling\_tf\_bert.TFBertForPreTrainingOutput](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.models.bert.modeling_tf_bert.TFBertForPreTrainingOutput)**or**tuple(tf.Tensor)

A [transformers.models.bert.modeling\_tf\_bert.TFBertForPreTrainingOutput](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.models.bert.modeling_tf_bert.TFBertForPreTrainingOutput) or a tuple of tf.Tensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **prediction\_logits** (tf.Tensor of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **seq\_relationship\_logits** (tf.Tensor of shape (batch\_size, 2)) — Prediction scores of the next sequence prediction (classification) head (scores of True/False continuation before SoftMax).
* **hidden\_states** (tuple(tf.Tensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of tf.Tensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [TFBertForPreTraining](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForPreTraining) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Examples:

Copied

>>> import tensorflow as tf

>>> from transformers import AutoTokenizer, TFBertForPreTraining

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = TFBertForPreTraining.from\_pretrained("bert-base-uncased")

>>> input\_ids = tokenizer("Hello, my dog is cute", add\_special\_tokens=True, return\_tensors="tf")

>>> *# Batch size 1*

>>> outputs = model(input\_ids)

>>> prediction\_logits, seq\_relationship\_logits = outputs[:2]

**TFBertModelLMHeadModel**

classtransformers.**TFBertLMHeadModel**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1485)

( \*args\*\*kwargs )

call

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1522)

( input\_ids: TFModelInputType | None = Noneattention\_mask: np.ndarray | tf.Tensor | None = Nonetoken\_type\_ids: np.ndarray | tf.Tensor | None = Noneposition\_ids: np.ndarray | tf.Tensor | None = Nonehead\_mask: np.ndarray | tf.Tensor | None = Noneinputs\_embeds: np.ndarray | tf.Tensor | None = Noneencoder\_hidden\_states: np.ndarray | tf.Tensor | None = Noneencoder\_attention\_mask: np.ndarray | tf.Tensor | None = Nonepast\_key\_values: Optional[Tuple[Tuple[Union[np.ndarray, tf.Tensor]]]] = Noneuse\_cache: Optional[bool] = Noneoutput\_attentions: Optional[bool] = Noneoutput\_hidden\_states: Optional[bool] = Nonereturn\_dict: Optional[bool] = Nonelabels: np.ndarray | tf.Tensor | None = Nonetraining: Optional[bool] = False\*\*kwargs ) **→** [transformers.modeling\_tf\_outputs.TFCausalLMOutputWithCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFCausalLMOutputWithCrossAttentions) or tuple(tf.Tensor)

Expand undefined parameters

**Returns**

[transformers.modeling\_tf\_outputs.TFCausalLMOutputWithCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFCausalLMOutputWithCrossAttentions)**or**tuple(tf.Tensor)

A [transformers.modeling\_tf\_outputs.TFCausalLMOutputWithCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFCausalLMOutputWithCrossAttentions) or a tuple of tf.Tensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (tf.Tensor of shape (n,), *optional*, where n is the number of non-masked labels, returned when labels is provided) — Language modeling loss (for next-token prediction).
* **logits** (tf.Tensor of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **hidden\_states** (tuple(tf.Tensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of tf.Tensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

* **cross\_attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights of the decoder’s cross-attention layer, after the attention softmax, used to compute the weighted average in the cross-attention heads.

* **past\_key\_values** (List[tf.Tensor], *optional*, returned when use\_cache=True is passed or when config.use\_cache=True) — List of tf.Tensor of length config.n\_layers, with each tensor of shape (2, batch\_size, num\_heads, sequence\_length, embed\_size\_per\_head)).

Contains pre-computed hidden-states (key and values in the attention blocks) that can be used (see past\_key\_values input) to speed up sequential decoding.

encoder\_hidden\_states (tf.Tensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*): Sequence of hidden-states at the output of the last layer of the encoder. Used in the cross-attention if the model is configured as a decoder. encoder\_attention\_mask (tf.Tensor of shape (batch\_size, sequence\_length), *optional*): Mask to avoid performing attention on the padding token indices of the encoder input. This mask is used in the cross-attention if the model is configured as a decoder. Mask values selected in [0, 1]:

* 1 for tokens that are **not masked**,
* 0 for tokens that are **masked**.

past\_key\_values (Tuple[Tuple[tf.Tensor]] of length config.n\_layers) contains precomputed key and value hidden states of the attention blocks. Can be used to speed up decoding. If past\_key\_values are used, the user can optionally input only the last decoder\_input\_ids (those that don’t have their past key value states given to this model) of shape (batch\_size, 1) instead of all decoder\_input\_ids of shape (batch\_size, sequence\_length). use\_cache (bool, *optional*, defaults to True): If set to True, past\_key\_values key value states are returned and can be used to speed up decoding (see past\_key\_values). Set to False during training, True during generation labels (tf.Tensor or np.ndarray of shape (batch\_size, sequence\_length), *optional*): Labels for computing the cross entropy classification loss. Indices should be in [0, ..., config.vocab\_size - 1].

Example:

Copied

>>> from transformers import AutoTokenizer, TFBertLMHeadModel

>>> import tensorflow as tf

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = TFBertLMHeadModel.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="tf")

>>> outputs = model(inputs)

>>> logits = outputs.logits

**TFBertForMaskedLM**

classtransformers.**TFBertForMaskedLM**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1389)

( \*args\*\*kwargs )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with a language modeling head on top.

This model inherits from [TFPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a [tf.keras.Model](https://www.tensorflow.org/api_docs/python/tf/keras/Model) subclass. Use it as a regular TF 2.0 Keras Model and refer to the TF 2.0 documentation for all matter related to general usage and behavior.

TensorFlow models and layers in transformers accept two formats as input:

* having all inputs as keyword arguments (like PyTorch models), or
* having all inputs as a list, tuple or dict in the first positional argument.

The reason the second format is supported is that Keras methods prefer this format when passing inputs to models and layers. Because of this support, when using methods like model.fit() things should “just work” for you - just pass your inputs and labels in any format that model.fit() supports! If, however, you want to use the second format outside of Keras methods like fit() and predict(), such as when creating your own layers or models with the Keras Functional API, there are three possibilities you can use to gather all the input Tensors in the first positional argument:

* a single Tensor with input\_ids only and nothing else: model(input\_ids)
* a list of varying length with one or several input Tensors IN THE ORDER given in the docstring: model([input\_ids, attention\_mask]) or model([input\_ids, attention\_mask, token\_type\_ids])
* a dictionary with one or several input Tensors associated to the input names given in the docstring: model({"input\_ids": input\_ids, "token\_type\_ids": token\_type\_ids})

Note that when creating models and layers with [subclassing](https://keras.io/guides/making_new_layers_and_models_via_subclassing/) then you don’t need to worry about any of this, as you can just pass inputs like you would to any other Python function!

call

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1417)

( input\_ids: TFModelInputType | None = Noneattention\_mask: np.ndarray | tf.Tensor | None = Nonetoken\_type\_ids: np.ndarray | tf.Tensor | None = Noneposition\_ids: np.ndarray | tf.Tensor | None = Nonehead\_mask: np.ndarray | tf.Tensor | None = Noneinputs\_embeds: np.ndarray | tf.Tensor | None = Noneoutput\_attentions: Optional[bool] = Noneoutput\_hidden\_states: Optional[bool] = Nonereturn\_dict: Optional[bool] = Nonelabels: np.ndarray | tf.Tensor | None = Nonetraining: Optional[bool] = False ) **→** [transformers.modeling\_tf\_outputs.TFMaskedLMOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFMaskedLMOutput) or tuple(tf.Tensor)

Expand 11 parameters

**Parameters**

* **input\_ids** (np.ndarray, tf.Tensor, List[tf.Tensor] `Dict[str, tf.Tensor] or Dict[str, np.ndarray] and each example must have the shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) and [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (np.ndarray or tf.Tensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple. This argument can be used in eager mode, in graph mode the value will always be set to True.
* **training** (bool, *optional*, defaults to `False“) — Whether or not to use the model in training mode (some modules like dropout modules have different behaviors between training and evaluation).
* **labels** (tf.Tensor or np.ndarray of shape (batch\_size, sequence\_length), *optional*) — Labels for computing the masked language modeling loss. Indices should be in [-100, 0, ..., config.vocab\_size] (see input\_ids docstring) Tokens with indices set to -100 are ignored (masked), the loss is only computed for the tokens with labels in [0, ..., config.vocab\_size]

**Returns**

[transformers.modeling\_tf\_outputs.TFMaskedLMOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFMaskedLMOutput)**or**tuple(tf.Tensor)

A [transformers.modeling\_tf\_outputs.TFMaskedLMOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFMaskedLMOutput) or a tuple of tf.Tensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (tf.Tensor of shape (n,), *optional*, where n is the number of non-masked labels, returned when labels is provided) — Masked language modeling (MLM) loss.
* **logits** (tf.Tensor of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **hidden\_states** (tuple(tf.Tensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of tf.Tensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [TFBertForMaskedLM](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForMaskedLM) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, TFBertForMaskedLM

>>> import tensorflow as tf

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = TFBertForMaskedLM.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("The capital of France is [MASK].", return\_tensors="tf")

>>> logits = model(\*\*inputs).logits

>>> *# retrieve index of [MASK]*

>>> mask\_token\_index = tf.where((inputs.input\_ids == tokenizer.mask\_token\_id)[0])

>>> selected\_logits = tf.gather\_nd(logits[0], indices=mask\_token\_index)

>>> predicted\_token\_id = tf.math.argmax(selected\_logits, axis=-1)

>>> tokenizer.decode(predicted\_token\_id)

'paris'

Copied

>>> labels = tokenizer("The capital of France is Paris.", return\_tensors="tf")["input\_ids"]

>>> *# mask labels of non-[MASK] tokens*

>>> labels = tf.where(inputs.input\_ids == tokenizer.mask\_token\_id, labels, -100)

>>> outputs = model(\*\*inputs, labels=labels)

>>> round(float(outputs.loss), 2)

0.88

**TFBertForNextSentencePrediction**

classtransformers.**TFBertForNextSentencePrediction**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1625)

( \*args\*\*kwargs )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with a next sentence prediction (classification) head on top.

This model inherits from [TFPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a [tf.keras.Model](https://www.tensorflow.org/api_docs/python/tf/keras/Model) subclass. Use it as a regular TF 2.0 Keras Model and refer to the TF 2.0 documentation for all matter related to general usage and behavior.

TensorFlow models and layers in transformers accept two formats as input:

* having all inputs as keyword arguments (like PyTorch models), or
* having all inputs as a list, tuple or dict in the first positional argument.

The reason the second format is supported is that Keras methods prefer this format when passing inputs to models and layers. Because of this support, when using methods like model.fit() things should “just work” for you - just pass your inputs and labels in any format that model.fit() supports! If, however, you want to use the second format outside of Keras methods like fit() and predict(), such as when creating your own layers or models with the Keras Functional API, there are three possibilities you can use to gather all the input Tensors in the first positional argument:

* a single Tensor with input\_ids only and nothing else: model(input\_ids)
* a list of varying length with one or several input Tensors IN THE ORDER given in the docstring: model([input\_ids, attention\_mask]) or model([input\_ids, attention\_mask, token\_type\_ids])
* a dictionary with one or several input Tensors associated to the input names given in the docstring: model({"input\_ids": input\_ids, "token\_type\_ids": token\_type\_ids})

Note that when creating models and layers with [subclassing](https://keras.io/guides/making_new_layers_and_models_via_subclassing/) then you don’t need to worry about any of this, as you can just pass inputs like you would to any other Python function!

call

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1635)

( input\_ids: TFModelInputType | None = Noneattention\_mask: np.ndarray | tf.Tensor | None = Nonetoken\_type\_ids: np.ndarray | tf.Tensor | None = Noneposition\_ids: np.ndarray | tf.Tensor | None = Nonehead\_mask: np.ndarray | tf.Tensor | None = Noneinputs\_embeds: np.ndarray | tf.Tensor | None = Noneoutput\_attentions: Optional[bool] = Noneoutput\_hidden\_states: Optional[bool] = Nonereturn\_dict: Optional[bool] = Nonenext\_sentence\_label: np.ndarray | tf.Tensor | None = Nonetraining: Optional[bool] = False ) **→** [transformers.modeling\_tf\_outputs.TFNextSentencePredictorOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFNextSentencePredictorOutput) or tuple(tf.Tensor)

Expand 10 parameters

**Parameters**

* **input\_ids** (np.ndarray, tf.Tensor, List[tf.Tensor] `Dict[str, tf.Tensor] or Dict[str, np.ndarray] and each example must have the shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) and [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (np.ndarray or tf.Tensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple. This argument can be used in eager mode, in graph mode the value will always be set to True.
* **training** (bool, *optional*, defaults to `False“) — Whether or not to use the model in training mode (some modules like dropout modules have different behaviors between training and evaluation).

**Returns**

[transformers.modeling\_tf\_outputs.TFNextSentencePredictorOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFNextSentencePredictorOutput)**or**tuple(tf.Tensor)

A [transformers.modeling\_tf\_outputs.TFNextSentencePredictorOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFNextSentencePredictorOutput) or a tuple of tf.Tensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (tf.Tensor of shape (n,), *optional*, where n is the number of non-masked labels, returned when next\_sentence\_label is provided) — Next sentence prediction loss.
* **logits** (tf.Tensor of shape (batch\_size, 2)) — Prediction scores of the next sequence prediction (classification) head (scores of True/False continuation before SoftMax).
* **hidden\_states** (tuple(tf.Tensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of tf.Tensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [TFBertForNextSentencePrediction](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForNextSentencePrediction) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Examples:

Copied

>>> import tensorflow as tf

>>> from transformers import AutoTokenizer, TFBertForNextSentencePrediction

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = TFBertForNextSentencePrediction.from\_pretrained("bert-base-uncased")

>>> prompt = "In Italy, pizza served in formal settings, such as at a restaurant, is presented unsliced."

>>> next\_sentence = "The sky is blue due to the shorter wavelength of blue light."

>>> encoding = tokenizer(prompt, next\_sentence, return\_tensors="tf")

>>> logits = model(encoding["input\_ids"], token\_type\_ids=encoding["token\_type\_ids"])[0]

>>> assert logits[0][0] < logits[0][1] *# the next sentence was random*

**TFBertForSequenceClassification**

classtransformers.**TFBertForSequenceClassification**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1721)

( \*args\*\*kwargs )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel.from_pretrained) method to load the model weights.

Bert Model transformer with a sequence classification/regression head on top (a linear layer on top of the pooled output) e.g. for GLUE tasks.

This model inherits from [TFPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a [tf.keras.Model](https://www.tensorflow.org/api_docs/python/tf/keras/Model) subclass. Use it as a regular TF 2.0 Keras Model and refer to the TF 2.0 documentation for all matter related to general usage and behavior.

TensorFlow models and layers in transformers accept two formats as input:

* having all inputs as keyword arguments (like PyTorch models), or
* having all inputs as a list, tuple or dict in the first positional argument.

The reason the second format is supported is that Keras methods prefer this format when passing inputs to models and layers. Because of this support, when using methods like model.fit() things should “just work” for you - just pass your inputs and labels in any format that model.fit() supports! If, however, you want to use the second format outside of Keras methods like fit() and predict(), such as when creating your own layers or models with the Keras Functional API, there are three possibilities you can use to gather all the input Tensors in the first positional argument:

* a single Tensor with input\_ids only and nothing else: model(input\_ids)
* a list of varying length with one or several input Tensors IN THE ORDER given in the docstring: model([input\_ids, attention\_mask]) or model([input\_ids, attention\_mask, token\_type\_ids])
* a dictionary with one or several input Tensors associated to the input names given in the docstring: model({"input\_ids": input\_ids, "token\_type\_ids": token\_type\_ids})

Note that when creating models and layers with [subclassing](https://keras.io/guides/making_new_layers_and_models_via_subclassing/) then you don’t need to worry about any of this, as you can just pass inputs like you would to any other Python function!

call

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1743)

( input\_ids: TFModelInputType | None = Noneattention\_mask: np.ndarray | tf.Tensor | None = Nonetoken\_type\_ids: np.ndarray | tf.Tensor | None = Noneposition\_ids: np.ndarray | tf.Tensor | None = Nonehead\_mask: np.ndarray | tf.Tensor | None = Noneinputs\_embeds: np.ndarray | tf.Tensor | None = Noneoutput\_attentions: Optional[bool] = Noneoutput\_hidden\_states: Optional[bool] = Nonereturn\_dict: Optional[bool] = Nonelabels: np.ndarray | tf.Tensor | None = Nonetraining: Optional[bool] = False ) **→** [transformers.modeling\_tf\_outputs.TFSequenceClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFSequenceClassifierOutput) or tuple(tf.Tensor)

Expand 11 parameters

**Parameters**

* **input\_ids** (np.ndarray, tf.Tensor, List[tf.Tensor] `Dict[str, tf.Tensor] or Dict[str, np.ndarray] and each example must have the shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) and [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (np.ndarray or tf.Tensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple. This argument can be used in eager mode, in graph mode the value will always be set to True.
* **training** (bool, *optional*, defaults to `False“) — Whether or not to use the model in training mode (some modules like dropout modules have different behaviors between training and evaluation).
* **labels** (tf.Tensor or np.ndarray of shape (batch\_size,), *optional*) — Labels for computing the sequence classification/regression loss. Indices should be in [0, ..., config.num\_labels - 1]. If config.num\_labels == 1 a regression loss is computed (Mean-Square loss), If config.num\_labels > 1 a classification loss is computed (Cross-Entropy).

**Returns**

[transformers.modeling\_tf\_outputs.TFSequenceClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFSequenceClassifierOutput)**or**tuple(tf.Tensor)

A [transformers.modeling\_tf\_outputs.TFSequenceClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFSequenceClassifierOutput) or a tuple of tf.Tensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (tf.Tensor of shape (batch\_size, ), *optional*, returned when labels is provided) — Classification (or regression if config.num\_labels==1) loss.
* **logits** (tf.Tensor of shape (batch\_size, config.num\_labels)) — Classification (or regression if config.num\_labels==1) scores (before SoftMax).
* **hidden\_states** (tuple(tf.Tensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of tf.Tensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [TFBertForSequenceClassification](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForSequenceClassification) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, TFBertForSequenceClassification

>>> import tensorflow as tf

>>> tokenizer = AutoTokenizer.from\_pretrained("ydshieh/bert-base-uncased-yelp-polarity")

>>> model = TFBertForSequenceClassification.from\_pretrained("ydshieh/bert-base-uncased-yelp-polarity")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="tf")

>>> logits = model(\*\*inputs).logits

>>> predicted\_class\_id = int(tf.math.argmax(logits, axis=-1)[0])

>>> model.config.id2label[predicted\_class\_id]

'LABEL\_1'

Copied

>>> *# To train a model on `num\_labels` classes, you can pass `num\_labels=num\_labels` to `.from\_pretrained(...)`*

>>> num\_labels = len(model.config.id2label)

>>> model = TFBertForSequenceClassification.from\_pretrained("ydshieh/bert-base-uncased-yelp-polarity", num\_labels=num\_labels)

>>> labels = tf.constant(1)

>>> loss = model(\*\*inputs, labels=labels).loss

>>> round(float(loss), 2)

0.01

**TFBertForMultipleChoice**

classtransformers.**TFBertForMultipleChoice**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1819)

( \*args\*\*kwargs )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with a multiple choice classification head on top (a linear layer on top of the pooled output and a softmax) e.g. for RocStories/SWAG tasks.

This model inherits from [TFPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a [tf.keras.Model](https://www.tensorflow.org/api_docs/python/tf/keras/Model) subclass. Use it as a regular TF 2.0 Keras Model and refer to the TF 2.0 documentation for all matter related to general usage and behavior.

TensorFlow models and layers in transformers accept two formats as input:

* having all inputs as keyword arguments (like PyTorch models), or
* having all inputs as a list, tuple or dict in the first positional argument.

The reason the second format is supported is that Keras methods prefer this format when passing inputs to models and layers. Because of this support, when using methods like model.fit() things should “just work” for you - just pass your inputs and labels in any format that model.fit() supports! If, however, you want to use the second format outside of Keras methods like fit() and predict(), such as when creating your own layers or models with the Keras Functional API, there are three possibilities you can use to gather all the input Tensors in the first positional argument:

* a single Tensor with input\_ids only and nothing else: model(input\_ids)
* a list of varying length with one or several input Tensors IN THE ORDER given in the docstring: model([input\_ids, attention\_mask]) or model([input\_ids, attention\_mask, token\_type\_ids])
* a dictionary with one or several input Tensors associated to the input names given in the docstring: model({"input\_ids": input\_ids, "token\_type\_ids": token\_type\_ids})

Note that when creating models and layers with [subclassing](https://keras.io/guides/making_new_layers_and_models_via_subclassing/) then you don’t need to worry about any of this, as you can just pass inputs like you would to any other Python function!

call

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1834)

( input\_ids: TFModelInputType | None = Noneattention\_mask: np.ndarray | tf.Tensor | None = Nonetoken\_type\_ids: np.ndarray | tf.Tensor | None = Noneposition\_ids: np.ndarray | tf.Tensor | None = Nonehead\_mask: np.ndarray | tf.Tensor | None = Noneinputs\_embeds: np.ndarray | tf.Tensor | None = Noneoutput\_attentions: Optional[bool] = Noneoutput\_hidden\_states: Optional[bool] = Nonereturn\_dict: Optional[bool] = Nonelabels: np.ndarray | tf.Tensor | None = Nonetraining: Optional[bool] = False ) **→** [transformers.modeling\_tf\_outputs.TFMultipleChoiceModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFMultipleChoiceModelOutput) or tuple(tf.Tensor)

Expand 11 parameters

**Parameters**

* **input\_ids** (np.ndarray, tf.Tensor, List[tf.Tensor] `Dict[str, tf.Tensor] or Dict[str, np.ndarray] and each example must have the shape (batch\_size, num\_choices, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) and [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (np.ndarray or tf.Tensor of shape (batch\_size, num\_choices, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, num\_choices, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, num\_choices, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (np.ndarray or tf.Tensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (np.ndarray or tf.Tensor of shape (batch\_size, num\_choices, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple. This argument can be used in eager mode, in graph mode the value will always be set to True.
* **training** (bool, *optional*, defaults to `False“) — Whether or not to use the model in training mode (some modules like dropout modules have different behaviors between training and evaluation).
* **labels** (tf.Tensor or np.ndarray of shape (batch\_size,), *optional*) — Labels for computing the multiple choice classification loss. Indices should be in [0, ..., num\_choices] where num\_choices is the size of the second dimension of the input tensors. (See input\_ids above)

**Returns**

[transformers.modeling\_tf\_outputs.TFMultipleChoiceModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFMultipleChoiceModelOutput)**or**tuple(tf.Tensor)

A [transformers.modeling\_tf\_outputs.TFMultipleChoiceModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFMultipleChoiceModelOutput) or a tuple of tf.Tensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (tf.Tensor of shape *(batch\_size, )*, *optional*, returned when labels is provided) — Classification loss.
* **logits** (tf.Tensor of shape (batch\_size, num\_choices)) — *num\_choices* is the second dimension of the input tensors. (see *input\_ids* above).

Classification scores (before SoftMax).

* **hidden\_states** (tuple(tf.Tensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of tf.Tensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [TFBertForMultipleChoice](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForMultipleChoice) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, TFBertForMultipleChoice

>>> import tensorflow as tf

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = TFBertForMultipleChoice.from\_pretrained("bert-base-uncased")

>>> prompt = "In Italy, pizza served in formal settings, such as at a restaurant, is presented unsliced."

>>> choice0 = "It is eaten with a fork and a knife."

>>> choice1 = "It is eaten while held in the hand."

>>> encoding = tokenizer([prompt, prompt], [choice0, choice1], return\_tensors="tf", padding=True)

>>> inputs = {k: tf.expand\_dims(v, 0) for k, v in encoding.items()}

>>> outputs = model(inputs) *# batch size is 1*

>>> *# the linear classifier still needs to be trained*

>>> logits = outputs.logits

**TFBertForTokenClassification**

classtransformers.**TFBertForTokenClassification**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1930)

( \*args\*\*kwargs )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with a token classification head on top (a linear layer on top of the hidden-states output) e.g. for Named-Entity-Recognition (NER) tasks.

This model inherits from [TFPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a [tf.keras.Model](https://www.tensorflow.org/api_docs/python/tf/keras/Model) subclass. Use it as a regular TF 2.0 Keras Model and refer to the TF 2.0 documentation for all matter related to general usage and behavior.

TensorFlow models and layers in transformers accept two formats as input:

* having all inputs as keyword arguments (like PyTorch models), or
* having all inputs as a list, tuple or dict in the first positional argument.

The reason the second format is supported is that Keras methods prefer this format when passing inputs to models and layers. Because of this support, when using methods like model.fit() things should “just work” for you - just pass your inputs and labels in any format that model.fit() supports! If, however, you want to use the second format outside of Keras methods like fit() and predict(), such as when creating your own layers or models with the Keras Functional API, there are three possibilities you can use to gather all the input Tensors in the first positional argument:

* a single Tensor with input\_ids only and nothing else: model(input\_ids)
* a list of varying length with one or several input Tensors IN THE ORDER given in the docstring: model([input\_ids, attention\_mask]) or model([input\_ids, attention\_mask, token\_type\_ids])
* a dictionary with one or several input Tensors associated to the input names given in the docstring: model({"input\_ids": input\_ids, "token\_type\_ids": token\_type\_ids})

Note that when creating models and layers with [subclassing](https://keras.io/guides/making_new_layers_and_models_via_subclassing/) then you don’t need to worry about any of this, as you can just pass inputs like you would to any other Python function!

call

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L1958)

( input\_ids: TFModelInputType | None = Noneattention\_mask: np.ndarray | tf.Tensor | None = Nonetoken\_type\_ids: np.ndarray | tf.Tensor | None = Noneposition\_ids: np.ndarray | tf.Tensor | None = Nonehead\_mask: np.ndarray | tf.Tensor | None = Noneinputs\_embeds: np.ndarray | tf.Tensor | None = Noneoutput\_attentions: Optional[bool] = Noneoutput\_hidden\_states: Optional[bool] = Nonereturn\_dict: Optional[bool] = Nonelabels: np.ndarray | tf.Tensor | None = Nonetraining: Optional[bool] = False ) **→** [transformers.modeling\_tf\_outputs.TFTokenClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFTokenClassifierOutput) or tuple(tf.Tensor)

Expand 11 parameters

**Parameters**

* **input\_ids** (np.ndarray, tf.Tensor, List[tf.Tensor] `Dict[str, tf.Tensor] or Dict[str, np.ndarray] and each example must have the shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) and [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (np.ndarray or tf.Tensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple. This argument can be used in eager mode, in graph mode the value will always be set to True.
* **training** (bool, *optional*, defaults to `False“) — Whether or not to use the model in training mode (some modules like dropout modules have different behaviors between training and evaluation).
* **labels** (tf.Tensor or np.ndarray of shape (batch\_size, sequence\_length), *optional*) — Labels for computing the token classification loss. Indices should be in [0, ..., config.num\_labels - 1].

**Returns**

[transformers.modeling\_tf\_outputs.TFTokenClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFTokenClassifierOutput)**or**tuple(tf.Tensor)

A [transformers.modeling\_tf\_outputs.TFTokenClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFTokenClassifierOutput) or a tuple of tf.Tensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (tf.Tensor of shape (n,), *optional*, where n is the number of unmasked labels, returned when labels is provided) — Classification loss.
* **logits** (tf.Tensor of shape (batch\_size, sequence\_length, config.num\_labels)) — Classification scores (before SoftMax).
* **hidden\_states** (tuple(tf.Tensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of tf.Tensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [TFBertForTokenClassification](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForTokenClassification) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, TFBertForTokenClassification

>>> import tensorflow as tf

>>> tokenizer = AutoTokenizer.from\_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")

>>> model = TFBertForTokenClassification.from\_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")

>>> inputs = tokenizer(

... "HuggingFace is a company based in Paris and New York", add\_special\_tokens=False, return\_tensors="tf"

... )

>>> logits = model(\*\*inputs).logits

>>> predicted\_token\_class\_ids = tf.math.argmax(logits, axis=-1)

>>> *# Note that tokens are classified rather then input words which means that*

>>> *# there might be more predicted token classes than words.*

>>> *# Multiple token classes might account for the same word*

>>> predicted\_tokens\_classes = [model.config.id2label[t] for t in predicted\_token\_class\_ids[0].numpy().tolist()]

>>> predicted\_tokens\_classes

['O', 'I-ORG', 'I-ORG', 'I-ORG', 'O', 'O', 'O', 'O', 'O', 'I-LOC', 'O', 'I-LOC', 'I-LOC']

Copied

>>> labels = predicted\_token\_class\_ids

>>> loss = tf.math.reduce\_mean(model(\*\*inputs, labels=labels).loss)

>>> round(float(loss), 2)

0.01

**TFBertForQuestionAnswering**

classtransformers.**TFBertForQuestionAnswering**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L2032)

( \*args\*\*kwargs )

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel.from_pretrained) method to load the model weights.

Bert Model with a span classification head on top for extractive question-answering tasks like SQuAD (a linear layer on top of the hidden-states output to compute span start logits and span end logits).

This model inherits from [TFPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.TFPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a [tf.keras.Model](https://www.tensorflow.org/api_docs/python/tf/keras/Model) subclass. Use it as a regular TF 2.0 Keras Model and refer to the TF 2.0 documentation for all matter related to general usage and behavior.

TensorFlow models and layers in transformers accept two formats as input:

* having all inputs as keyword arguments (like PyTorch models), or
* having all inputs as a list, tuple or dict in the first positional argument.

The reason the second format is supported is that Keras methods prefer this format when passing inputs to models and layers. Because of this support, when using methods like model.fit() things should “just work” for you - just pass your inputs and labels in any format that model.fit() supports! If, however, you want to use the second format outside of Keras methods like fit() and predict(), such as when creating your own layers or models with the Keras Functional API, there are three possibilities you can use to gather all the input Tensors in the first positional argument:

* a single Tensor with input\_ids only and nothing else: model(input\_ids)
* a list of varying length with one or several input Tensors IN THE ORDER given in the docstring: model([input\_ids, attention\_mask]) or model([input\_ids, attention\_mask, token\_type\_ids])
* a dictionary with one or several input Tensors associated to the input names given in the docstring: model({"input\_ids": input\_ids, "token\_type\_ids": token\_type\_ids})

Note that when creating models and layers with [subclassing](https://keras.io/guides/making_new_layers_and_models_via_subclassing/) then you don’t need to worry about any of this, as you can just pass inputs like you would to any other Python function!

call

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_tf_bert.py#L2055)

( input\_ids: TFModelInputType | None = Noneattention\_mask: np.ndarray | tf.Tensor | None = Nonetoken\_type\_ids: np.ndarray | tf.Tensor | None = Noneposition\_ids: np.ndarray | tf.Tensor | None = Nonehead\_mask: np.ndarray | tf.Tensor | None = Noneinputs\_embeds: np.ndarray | tf.Tensor | None = Noneoutput\_attentions: Optional[bool] = Noneoutput\_hidden\_states: Optional[bool] = Nonereturn\_dict: Optional[bool] = Nonestart\_positions: np.ndarray | tf.Tensor | None = Noneend\_positions: np.ndarray | tf.Tensor | None = Nonetraining: Optional[bool] = False ) **→** [transformers.modeling\_tf\_outputs.TFQuestionAnsweringModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFQuestionAnsweringModelOutput) or tuple(tf.Tensor)

Expand 12 parameters

**Parameters**

* **input\_ids** (np.ndarray, tf.Tensor, List[tf.Tensor] `Dict[str, tf.Tensor] or Dict[str, np.ndarray] and each example must have the shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) and [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **head\_mask** (np.ndarray or tf.Tensor of shape (num\_heads,) or (num\_layers, num\_heads), *optional*) — Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **inputs\_embeds** (np.ndarray or tf.Tensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail. This argument can be used only in eager mode, in graph mode the value in the config will be used instead.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple. This argument can be used in eager mode, in graph mode the value will always be set to True.
* **training** (bool, *optional*, defaults to `False“) — Whether or not to use the model in training mode (some modules like dropout modules have different behaviors between training and evaluation).
* **start\_positions** (tf.Tensor or np.ndarray of shape (batch\_size,), *optional*) — Labels for position (index) of the start of the labelled span for computing the token classification loss. Positions are clamped to the length of the sequence (sequence\_length). Position outside of the sequence are not taken into account for computing the loss.
* **end\_positions** (tf.Tensor or np.ndarray of shape (batch\_size,), *optional*) — Labels for position (index) of the end of the labelled span for computing the token classification loss. Positions are clamped to the length of the sequence (sequence\_length). Position outside of the sequence are not taken into account for computing the loss.

**Returns**

[transformers.modeling\_tf\_outputs.TFQuestionAnsweringModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFQuestionAnsweringModelOutput)**or**tuple(tf.Tensor)

A [transformers.modeling\_tf\_outputs.TFQuestionAnsweringModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_tf_outputs.TFQuestionAnsweringModelOutput) or a tuple of tf.Tensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **loss** (tf.Tensor of shape (batch\_size, ), *optional*, returned when start\_positions and end\_positions are provided) — Total span extraction loss is the sum of a Cross-Entropy for the start and end positions.
* **start\_logits** (tf.Tensor of shape (batch\_size, sequence\_length)) — Span-start scores (before SoftMax).
* **end\_logits** (tf.Tensor of shape (batch\_size, sequence\_length)) — Span-end scores (before SoftMax).
* **hidden\_states** (tuple(tf.Tensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of tf.Tensor (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(tf.Tensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of tf.Tensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [TFBertForQuestionAnswering](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.TFBertForQuestionAnswering) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, TFBertForQuestionAnswering

>>> import tensorflow as tf

>>> tokenizer = AutoTokenizer.from\_pretrained("ydshieh/bert-base-cased-squad2")

>>> model = TFBertForQuestionAnswering.from\_pretrained("ydshieh/bert-base-cased-squad2")

>>> question, text = "Who was Jim Henson?", "Jim Henson was a nice puppet"

>>> inputs = tokenizer(question, text, return\_tensors="tf")

>>> outputs = model(\*\*inputs)

>>> answer\_start\_index = int(tf.math.argmax(outputs.start\_logits, axis=-1)[0])

>>> answer\_end\_index = int(tf.math.argmax(outputs.end\_logits, axis=-1)[0])

>>> predict\_answer\_tokens = inputs.input\_ids[0, answer\_start\_index : answer\_end\_index + 1]

>>> tokenizer.decode(predict\_answer\_tokens)

'a nice puppet'

Copied

>>> *# target is "nice puppet"*

>>> target\_start\_index = tf.constant([14])

>>> target\_end\_index = tf.constant([15])

>>> outputs = model(\*\*inputs, start\_positions=target\_start\_index, end\_positions=target\_end\_index)

>>> loss = tf.math.reduce\_mean(outputs.loss)

>>> round(float(loss), 2)

7.41

JAX

Hide JAX content

**FlaxBertModel**

classtransformers.**FlaxBertModel**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L1030)

( config: BertConfiginput\_shape: typing.Tuple = (1, 1)seed: int = 0dtype: dtype = <class 'jax.numpy.float32'>\_do\_init: bool = Truegradient\_checkpointing: bool = False\*\*kwargs )

Expand 3 parameters

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.from_pretrained) method to load the model weights.
* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

The bare Bert Model transformer outputting raw hidden-states without any specific head on top.

This model inherits from [FlaxPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading, saving and converting weights from PyTorch models)

This model is also a [flax.linen.Module](https://flax.readthedocs.io/en/latest/api_reference/flax.linen/module.html) subclass. Use it as a regular Flax linen Module and refer to the Flax documentation for all matter related to general usage and behavior.

Finally, this model supports inherent JAX features such as:

* [Just-In-Time (JIT) compilation](https://jax.readthedocs.io/en/latest/jax.html#just-in-time-compilation-jit)
* [Automatic Differentiation](https://jax.readthedocs.io/en/latest/jax.html#automatic-differentiation)
* [Vectorization](https://jax.readthedocs.io/en/latest/jax.html#vectorization-vmap)
* [Parallelization](https://jax.readthedocs.io/en/latest/jax.html#parallelization-pmap)

\_\_call\_\_

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L857)

( input\_idsattention\_mask = Nonetoken\_type\_ids = Noneposition\_ids = Nonehead\_mask = Noneencoder\_hidden\_states = Noneencoder\_attention\_mask = Noneparams: dict = Nonedropout\_rng: PRNGKey = Nonetrain: bool = Falseoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = Nonepast\_key\_values: dict = None ) **→** [transformers.modeling\_flax\_outputs.FlaxBaseModelOutputWithPooling](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxBaseModelOutputWithPooling) or tuple(torch.FloatTensor)

Expand 6 parameters

**Parameters**

* **input\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].
* **head\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), optional) -- Mask to nullify selected heads of the attention modules. Mask values selected in [0, 1]`:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.

**Returns**

[transformers.modeling\_flax\_outputs.FlaxBaseModelOutputWithPooling](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxBaseModelOutputWithPooling)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_flax\_outputs.FlaxBaseModelOutputWithPooling](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxBaseModelOutputWithPooling) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **last\_hidden\_state** (jnp.ndarray of shape (batch\_size, sequence\_length, hidden\_size)) — Sequence of hidden-states at the output of the last layer of the model.
* **pooler\_output** (jnp.ndarray of shape (batch\_size, hidden\_size)) — Last layer hidden-state of the first token of the sequence (classification token) further processed by a Linear layer and a Tanh activation function. The Linear layer weights are trained from the next sentence prediction (classification) objective during pretraining.
* **hidden\_states** (tuple(jnp.ndarray), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of jnp.ndarray (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(jnp.ndarray), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of jnp.ndarray (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The FlaxBertPreTrainedModel forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, FlaxBertModel

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = FlaxBertModel.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="jax")

>>> outputs = model(\*\*inputs)

>>> last\_hidden\_states = outputs.last\_hidden\_state

**FlaxBertForPreTraining**

classtransformers.**FlaxBertForPreTraining**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L1105)

( config: BertConfiginput\_shape: typing.Tuple = (1, 1)seed: int = 0dtype: dtype = <class 'jax.numpy.float32'>\_do\_init: bool = Truegradient\_checkpointing: bool = False\*\*kwargs )

Expand 3 parameters

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.from_pretrained) method to load the model weights.
* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

Bert Model with two heads on top as done during the pretraining: a masked language modeling head and a next sentence prediction (classification) head.

This model inherits from [FlaxPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading, saving and converting weights from PyTorch models)

This model is also a [flax.linen.Module](https://flax.readthedocs.io/en/latest/api_reference/flax.linen/module.html) subclass. Use it as a regular Flax linen Module and refer to the Flax documentation for all matter related to general usage and behavior.

Finally, this model supports inherent JAX features such as:

* [Just-In-Time (JIT) compilation](https://jax.readthedocs.io/en/latest/jax.html#just-in-time-compilation-jit)
* [Automatic Differentiation](https://jax.readthedocs.io/en/latest/jax.html#automatic-differentiation)
* [Vectorization](https://jax.readthedocs.io/en/latest/jax.html#vectorization-vmap)
* [Parallelization](https://jax.readthedocs.io/en/latest/jax.html#parallelization-pmap)

\_\_call\_\_

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L857)

( input\_idsattention\_mask = Nonetoken\_type\_ids = Noneposition\_ids = Nonehead\_mask = Noneencoder\_hidden\_states = Noneencoder\_attention\_mask = Noneparams: dict = Nonedropout\_rng: PRNGKey = Nonetrain: bool = Falseoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = Nonepast\_key\_values: dict = None ) **→** [transformers.models.bert.modeling\_flax\_bert.FlaxBertForPreTrainingOutput](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.models.bert.modeling_flax_bert.FlaxBertForPreTrainingOutput) or tuple(torch.FloatTensor)

Expand 6 parameters

**Parameters**

* **input\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].
* **head\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), optional) -- Mask to nullify selected heads of the attention modules. Mask values selected in [0, 1]`:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.

**Returns**

[transformers.models.bert.modeling\_flax\_bert.FlaxBertForPreTrainingOutput](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.models.bert.modeling_flax_bert.FlaxBertForPreTrainingOutput)**or**tuple(torch.FloatTensor)

A [transformers.models.bert.modeling\_flax\_bert.FlaxBertForPreTrainingOutput](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.models.bert.modeling_flax_bert.FlaxBertForPreTrainingOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **prediction\_logits** (jnp.ndarray of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **seq\_relationship\_logits** (jnp.ndarray of shape (batch\_size, 2)) — Prediction scores of the next sequence prediction (classification) head (scores of True/False continuation before SoftMax).
* **hidden\_states** (tuple(jnp.ndarray), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of jnp.ndarray (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(jnp.ndarray), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of jnp.ndarray (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The FlaxBertPreTrainedModel forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, FlaxBertForPreTraining

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = FlaxBertForPreTraining.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="np")

>>> outputs = model(\*\*inputs)

>>> prediction\_logits = outputs.prediction\_logits

>>> seq\_relationship\_logits = outputs.seq\_relationship\_logits

**FlaxBertForCausalLM**

classtransformers.**FlaxBertForCausalLM**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L1678)

( config: BertConfiginput\_shape: typing.Tuple = (1, 1)seed: int = 0dtype: dtype = <class 'jax.numpy.float32'>\_do\_init: bool = Truegradient\_checkpointing: bool = False\*\*kwargs )

Expand 3 parameters

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.from_pretrained) method to load the model weights.
* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

Bert Model with a language modeling head on top (a linear layer on top of the hidden-states output) e.g for autoregressive tasks.

This model inherits from [FlaxPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading, saving and converting weights from PyTorch models)

This model is also a [flax.linen.Module](https://flax.readthedocs.io/en/latest/api_reference/flax.linen/module.html) subclass. Use it as a regular Flax linen Module and refer to the Flax documentation for all matter related to general usage and behavior.

Finally, this model supports inherent JAX features such as:

* [Just-In-Time (JIT) compilation](https://jax.readthedocs.io/en/latest/jax.html#just-in-time-compilation-jit)
* [Automatic Differentiation](https://jax.readthedocs.io/en/latest/jax.html#automatic-differentiation)
* [Vectorization](https://jax.readthedocs.io/en/latest/jax.html#vectorization-vmap)
* [Parallelization](https://jax.readthedocs.io/en/latest/jax.html#parallelization-pmap)

\_\_call\_\_

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L857)

( input\_idsattention\_mask = Nonetoken\_type\_ids = Noneposition\_ids = Nonehead\_mask = Noneencoder\_hidden\_states = Noneencoder\_attention\_mask = Noneparams: dict = Nonedropout\_rng: PRNGKey = Nonetrain: bool = Falseoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = Nonepast\_key\_values: dict = None ) **→** [transformers.modeling\_flax\_outputs.FlaxCausalLMOutputWithCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxCausalLMOutputWithCrossAttentions) or tuple(torch.FloatTensor)

Expand 6 parameters

**Parameters**

* **input\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].
* **head\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), optional) -- Mask to nullify selected heads of the attention modules. Mask values selected in [0, 1]`:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.

**Returns**

[transformers.modeling\_flax\_outputs.FlaxCausalLMOutputWithCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxCausalLMOutputWithCrossAttentions)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_flax\_outputs.FlaxCausalLMOutputWithCrossAttentions](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxCausalLMOutputWithCrossAttentions) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **logits** (jnp.ndarray of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **hidden\_states** (tuple(jnp.ndarray), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of jnp.ndarray (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(jnp.ndarray), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of jnp.ndarray (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

* **cross\_attentions** (tuple(jnp.ndarray), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of jnp.ndarray (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Cross attentions weights after the attention softmax, used to compute the weighted average in the cross-attention heads.

* **past\_key\_values** (tuple(tuple(jnp.ndarray)), *optional*, returned when use\_cache=True is passed or when config.use\_cache=True) — Tuple of jnp.ndarray tuples of length config.n\_layers, with each tuple containing the cached key, value states of the self-attention and the cross-attention layers if model is used in encoder-decoder setting. Only relevant if config.is\_decoder = True.

Contains pre-computed hidden-states (key and values in the attention blocks) that can be used (see past\_key\_values input) to speed up sequential decoding.

The FlaxBertPreTrainedModel forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, FlaxBertForCausalLM

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = FlaxBertForCausalLM.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="np")

>>> outputs = model(\*\*inputs)

>>> *# retrieve logts for next token*

>>> next\_token\_logits = outputs.logits[:, -1]

**FlaxBertForMaskedLM**

classtransformers.**FlaxBertForMaskedLM**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L1196)

( config: BertConfiginput\_shape: typing.Tuple = (1, 1)seed: int = 0dtype: dtype = <class 'jax.numpy.float32'>\_do\_init: bool = Truegradient\_checkpointing: bool = False\*\*kwargs )

Expand 3 parameters

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.from_pretrained) method to load the model weights.
* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

Bert Model with a language modeling head on top.

This model inherits from [FlaxPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading, saving and converting weights from PyTorch models)

This model is also a [flax.linen.Module](https://flax.readthedocs.io/en/latest/api_reference/flax.linen/module.html) subclass. Use it as a regular Flax linen Module and refer to the Flax documentation for all matter related to general usage and behavior.

Finally, this model supports inherent JAX features such as:

* [Just-In-Time (JIT) compilation](https://jax.readthedocs.io/en/latest/jax.html#just-in-time-compilation-jit)
* [Automatic Differentiation](https://jax.readthedocs.io/en/latest/jax.html#automatic-differentiation)
* [Vectorization](https://jax.readthedocs.io/en/latest/jax.html#vectorization-vmap)
* [Parallelization](https://jax.readthedocs.io/en/latest/jax.html#parallelization-pmap)

\_\_call\_\_

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L857)

( input\_idsattention\_mask = Nonetoken\_type\_ids = Noneposition\_ids = Nonehead\_mask = Noneencoder\_hidden\_states = Noneencoder\_attention\_mask = Noneparams: dict = Nonedropout\_rng: PRNGKey = Nonetrain: bool = Falseoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = Nonepast\_key\_values: dict = None ) **→** [transformers.modeling\_flax\_outputs.FlaxMaskedLMOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxMaskedLMOutput) or tuple(torch.FloatTensor)

Expand 6 parameters

**Parameters**

* **input\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].
* **head\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), optional) -- Mask to nullify selected heads of the attention modules. Mask values selected in [0, 1]`:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.

**Returns**

[transformers.modeling\_flax\_outputs.FlaxMaskedLMOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxMaskedLMOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_flax\_outputs.FlaxMaskedLMOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxMaskedLMOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **logits** (jnp.ndarray of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **hidden\_states** (tuple(jnp.ndarray), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of jnp.ndarray (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(jnp.ndarray), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of jnp.ndarray (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The FlaxBertPreTrainedModel forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, FlaxBertForMaskedLM

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = FlaxBertForMaskedLM.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("The capital of France is [MASK].", return\_tensors="jax")

>>> outputs = model(\*\*inputs)

>>> logits = outputs.logits

**FlaxBertForNextSentencePrediction**

classtransformers.**FlaxBertForNextSentencePrediction**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L1260)

( config: BertConfiginput\_shape: typing.Tuple = (1, 1)seed: int = 0dtype: dtype = <class 'jax.numpy.float32'>\_do\_init: bool = Truegradient\_checkpointing: bool = False\*\*kwargs )

Expand 3 parameters

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.from_pretrained) method to load the model weights.
* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

Bert Model with a next sentence prediction (classification) head on top.

This model inherits from [FlaxPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading, saving and converting weights from PyTorch models)

This model is also a [flax.linen.Module](https://flax.readthedocs.io/en/latest/api_reference/flax.linen/module.html) subclass. Use it as a regular Flax linen Module and refer to the Flax documentation for all matter related to general usage and behavior.

Finally, this model supports inherent JAX features such as:

* [Just-In-Time (JIT) compilation](https://jax.readthedocs.io/en/latest/jax.html#just-in-time-compilation-jit)
* [Automatic Differentiation](https://jax.readthedocs.io/en/latest/jax.html#automatic-differentiation)
* [Vectorization](https://jax.readthedocs.io/en/latest/jax.html#vectorization-vmap)
* [Parallelization](https://jax.readthedocs.io/en/latest/jax.html#parallelization-pmap)

\_\_call\_\_

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L857)

( input\_idsattention\_mask = Nonetoken\_type\_ids = Noneposition\_ids = Nonehead\_mask = Noneencoder\_hidden\_states = Noneencoder\_attention\_mask = Noneparams: dict = Nonedropout\_rng: PRNGKey = Nonetrain: bool = Falseoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = Nonepast\_key\_values: dict = None ) **→** [transformers.modeling\_flax\_outputs.FlaxNextSentencePredictorOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxNextSentencePredictorOutput) or tuple(torch.FloatTensor)

Expand 6 parameters

**Parameters**

* **input\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].
* **head\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), optional) -- Mask to nullify selected heads of the attention modules. Mask values selected in [0, 1]`:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.

**Returns**

[transformers.modeling\_flax\_outputs.FlaxNextSentencePredictorOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxNextSentencePredictorOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_flax\_outputs.FlaxNextSentencePredictorOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxNextSentencePredictorOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **logits** (jnp.ndarray of shape (batch\_size, 2)) — Prediction scores of the next sequence prediction (classification) head (scores of True/False continuation before SoftMax).
* **hidden\_states** (tuple(jnp.ndarray), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of jnp.ndarray (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(jnp.ndarray), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of jnp.ndarray (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The FlaxBertPreTrainedModel forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, FlaxBertForNextSentencePrediction

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = FlaxBertForNextSentencePrediction.from\_pretrained("bert-base-uncased")

>>> prompt = "In Italy, pizza served in formal settings, such as at a restaurant, is presented unsliced."

>>> next\_sentence = "The sky is blue due to the shorter wavelength of blue light."

>>> encoding = tokenizer(prompt, next\_sentence, return\_tensors="jax")

>>> outputs = model(\*\*encoding)

>>> logits = outputs.logits

>>> assert logits[0, 0] < logits[0, 1] *# next sentence was random*

**FlaxBertForSequenceClassification**

classtransformers.**FlaxBertForSequenceClassification**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L1363)

( config: BertConfiginput\_shape: typing.Tuple = (1, 1)seed: int = 0dtype: dtype = <class 'jax.numpy.float32'>\_do\_init: bool = Truegradient\_checkpointing: bool = False\*\*kwargs )

Expand 3 parameters

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.from_pretrained) method to load the model weights.
* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

Bert Model transformer with a sequence classification/regression head on top (a linear layer on top of the pooled output) e.g. for GLUE tasks.

This model inherits from [FlaxPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading, saving and converting weights from PyTorch models)

This model is also a [flax.linen.Module](https://flax.readthedocs.io/en/latest/api_reference/flax.linen/module.html) subclass. Use it as a regular Flax linen Module and refer to the Flax documentation for all matter related to general usage and behavior.

Finally, this model supports inherent JAX features such as:

* [Just-In-Time (JIT) compilation](https://jax.readthedocs.io/en/latest/jax.html#just-in-time-compilation-jit)
* [Automatic Differentiation](https://jax.readthedocs.io/en/latest/jax.html#automatic-differentiation)
* [Vectorization](https://jax.readthedocs.io/en/latest/jax.html#vectorization-vmap)
* [Parallelization](https://jax.readthedocs.io/en/latest/jax.html#parallelization-pmap)

\_\_call\_\_

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L857)

( input\_idsattention\_mask = Nonetoken\_type\_ids = Noneposition\_ids = Nonehead\_mask = Noneencoder\_hidden\_states = Noneencoder\_attention\_mask = Noneparams: dict = Nonedropout\_rng: PRNGKey = Nonetrain: bool = Falseoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = Nonepast\_key\_values: dict = None ) **→** [transformers.modeling\_flax\_outputs.FlaxSequenceClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxSequenceClassifierOutput) or tuple(torch.FloatTensor)

Expand 6 parameters

**Parameters**

* **input\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].
* **head\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), optional) -- Mask to nullify selected heads of the attention modules. Mask values selected in [0, 1]`:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.

**Returns**

[transformers.modeling\_flax\_outputs.FlaxSequenceClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxSequenceClassifierOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_flax\_outputs.FlaxSequenceClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxSequenceClassifierOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **logits** (jnp.ndarray of shape (batch\_size, config.num\_labels)) — Classification (or regression if config.num\_labels==1) scores (before SoftMax).
* **hidden\_states** (tuple(jnp.ndarray), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of jnp.ndarray (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(jnp.ndarray), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of jnp.ndarray (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The FlaxBertPreTrainedModel forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, FlaxBertForSequenceClassification

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = FlaxBertForSequenceClassification.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="jax")

>>> outputs = model(\*\*inputs)

>>> logits = outputs.logits

**FlaxBertForMultipleChoice**

classtransformers.**FlaxBertForMultipleChoice**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L1443)

( config: BertConfiginput\_shape: typing.Tuple = (1, 1)seed: int = 0dtype: dtype = <class 'jax.numpy.float32'>\_do\_init: bool = Truegradient\_checkpointing: bool = False\*\*kwargs )

Expand 3 parameters

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.from_pretrained) method to load the model weights.
* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

Bert Model with a multiple choice classification head on top (a linear layer on top of the pooled output and a softmax) e.g. for RocStories/SWAG tasks.

This model inherits from [FlaxPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading, saving and converting weights from PyTorch models)

This model is also a [flax.linen.Module](https://flax.readthedocs.io/en/latest/api_reference/flax.linen/module.html) subclass. Use it as a regular Flax linen Module and refer to the Flax documentation for all matter related to general usage and behavior.

Finally, this model supports inherent JAX features such as:

* [Just-In-Time (JIT) compilation](https://jax.readthedocs.io/en/latest/jax.html#just-in-time-compilation-jit)
* [Automatic Differentiation](https://jax.readthedocs.io/en/latest/jax.html#automatic-differentiation)
* [Vectorization](https://jax.readthedocs.io/en/latest/jax.html#vectorization-vmap)
* [Parallelization](https://jax.readthedocs.io/en/latest/jax.html#parallelization-pmap)

\_\_call\_\_

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L857)

( input\_idsattention\_mask = Nonetoken\_type\_ids = Noneposition\_ids = Nonehead\_mask = Noneencoder\_hidden\_states = Noneencoder\_attention\_mask = Noneparams: dict = Nonedropout\_rng: PRNGKey = Nonetrain: bool = Falseoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = Nonepast\_key\_values: dict = None ) **→** [transformers.modeling\_flax\_outputs.FlaxMultipleChoiceModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxMultipleChoiceModelOutput) or tuple(torch.FloatTensor)

Expand 6 parameters

**Parameters**

* **input\_ids** (numpy.ndarray of shape (batch\_size, num\_choices, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (numpy.ndarray of shape (batch\_size, num\_choices, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (numpy.ndarray of shape (batch\_size, num\_choices, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (numpy.ndarray of shape (batch\_size, num\_choices, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].
* **head\_mask** (numpy.ndarray of shape (batch\_size, num\_choices, sequence\_length), optional) -- Mask to nullify selected heads of the attention modules. Mask values selected in [0, 1]`:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.

**Returns**

[transformers.modeling\_flax\_outputs.FlaxMultipleChoiceModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxMultipleChoiceModelOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_flax\_outputs.FlaxMultipleChoiceModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxMultipleChoiceModelOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **logits** (jnp.ndarray of shape (batch\_size, num\_choices)) — *num\_choices* is the second dimension of the input tensors. (see *input\_ids* above).

Classification scores (before SoftMax).

* **hidden\_states** (tuple(jnp.ndarray), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of jnp.ndarray (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(jnp.ndarray), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of jnp.ndarray (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The FlaxBertPreTrainedModel forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, FlaxBertForMultipleChoice

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = FlaxBertForMultipleChoice.from\_pretrained("bert-base-uncased")

>>> prompt = "In Italy, pizza served in formal settings, such as at a restaurant, is presented unsliced."

>>> choice0 = "It is eaten with a fork and a knife."

>>> choice1 = "It is eaten while held in the hand."

>>> encoding = tokenizer([prompt, prompt], [choice0, choice1], return\_tensors="jax", padding=True)

>>> outputs = model(\*\*{k: v[None, :] for k, v in encoding.items()})

>>> logits = outputs.logits

**FlaxBertForTokenClassification**

classtransformers.**FlaxBertForTokenClassification**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L1521)

( config: BertConfiginput\_shape: typing.Tuple = (1, 1)seed: int = 0dtype: dtype = <class 'jax.numpy.float32'>\_do\_init: bool = Truegradient\_checkpointing: bool = False\*\*kwargs )

Expand 3 parameters

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.from_pretrained) method to load the model weights.
* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

Bert Model with a token classification head on top (a linear layer on top of the hidden-states output) e.g. for Named-Entity-Recognition (NER) tasks.

This model inherits from [FlaxPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading, saving and converting weights from PyTorch models)

This model is also a [flax.linen.Module](https://flax.readthedocs.io/en/latest/api_reference/flax.linen/module.html) subclass. Use it as a regular Flax linen Module and refer to the Flax documentation for all matter related to general usage and behavior.

Finally, this model supports inherent JAX features such as:

* [Just-In-Time (JIT) compilation](https://jax.readthedocs.io/en/latest/jax.html#just-in-time-compilation-jit)
* [Automatic Differentiation](https://jax.readthedocs.io/en/latest/jax.html#automatic-differentiation)
* [Vectorization](https://jax.readthedocs.io/en/latest/jax.html#vectorization-vmap)
* [Parallelization](https://jax.readthedocs.io/en/latest/jax.html#parallelization-pmap)

\_\_call\_\_

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L857)

( input\_idsattention\_mask = Nonetoken\_type\_ids = Noneposition\_ids = Nonehead\_mask = Noneencoder\_hidden\_states = Noneencoder\_attention\_mask = Noneparams: dict = Nonedropout\_rng: PRNGKey = Nonetrain: bool = Falseoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = Nonepast\_key\_values: dict = None ) **→** [transformers.modeling\_flax\_outputs.FlaxTokenClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxTokenClassifierOutput) or tuple(torch.FloatTensor)

Expand 6 parameters

**Parameters**

* **input\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].
* **head\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), optional) -- Mask to nullify selected heads of the attention modules. Mask values selected in [0, 1]`:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.

**Returns**

[transformers.modeling\_flax\_outputs.FlaxTokenClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxTokenClassifierOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_flax\_outputs.FlaxTokenClassifierOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxTokenClassifierOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **logits** (jnp.ndarray of shape (batch\_size, sequence\_length, config.num\_labels)) — Classification scores (before SoftMax).
* **hidden\_states** (tuple(jnp.ndarray), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of jnp.ndarray (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(jnp.ndarray), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of jnp.ndarray (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The FlaxBertPreTrainedModel forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, FlaxBertForTokenClassification

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = FlaxBertForTokenClassification.from\_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello, my dog is cute", return\_tensors="jax")

>>> outputs = model(\*\*inputs)

>>> logits = outputs.logits

**FlaxBertForQuestionAnswering**

classtransformers.**FlaxBertForQuestionAnswering**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L1594)

( config: BertConfiginput\_shape: typing.Tuple = (1, 1)seed: int = 0dtype: dtype = <class 'jax.numpy.float32'>\_do\_init: bool = Truegradient\_checkpointing: bool = False\*\*kwargs )

Expand 3 parameters

**Parameters**

* **config** ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.from_pretrained) method to load the model weights.
* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

* **dtype** (jax.numpy.dtype, *optional*, defaults to jax.numpy.float32) — The data type of the computation. Can be one of jax.numpy.float32, jax.numpy.float16 (on GPUs) and jax.numpy.bfloat16 (on TPUs).

This can be used to enable mixed-precision training or half-precision inference on GPUs or TPUs. If specified all the computation will be performed with the given dtype.

**Note that this only specifies the dtype of the computation and does not influence the dtype of model parameters.**

If you wish to change the dtype of the model parameters, see [to\_fp16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_fp16) and [to\_bf16()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel.to_bf16).

Bert Model with a span classification head on top for extractive question-answering tasks like SQuAD (a linear layers on top of the hidden-states output to compute span start logits and span end logits).

This model inherits from [FlaxPreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.FlaxPreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading, saving and converting weights from PyTorch models)

This model is also a [flax.linen.Module](https://flax.readthedocs.io/en/latest/api_reference/flax.linen/module.html) subclass. Use it as a regular Flax linen Module and refer to the Flax documentation for all matter related to general usage and behavior.

Finally, this model supports inherent JAX features such as:

* [Just-In-Time (JIT) compilation](https://jax.readthedocs.io/en/latest/jax.html#just-in-time-compilation-jit)
* [Automatic Differentiation](https://jax.readthedocs.io/en/latest/jax.html#automatic-differentiation)
* [Vectorization](https://jax.readthedocs.io/en/latest/jax.html#vectorization-vmap)
* [Parallelization](https://jax.readthedocs.io/en/latest/jax.html#parallelization-pmap)

\_\_call\_\_

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_flax_bert.py#L857)

( input\_idsattention\_mask = Nonetoken\_type\_ids = Noneposition\_ids = Nonehead\_mask = Noneencoder\_hidden\_states = Noneencoder\_attention\_mask = Noneparams: dict = Nonedropout\_rng: PRNGKey = Nonetrain: bool = Falseoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = Nonepast\_key\_values: dict = None ) **→** [transformers.modeling\_flax\_outputs.FlaxQuestionAnsweringModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxQuestionAnsweringModelOutput) or tuple(torch.FloatTensor)

Expand 6 parameters

**Parameters**

* **input\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

* **token\_type\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1]:
  + 0 corresponds to a *sentence A* token,
  + 1 corresponds to a *sentence B* token.

[What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

* **position\_ids** (numpy.ndarray of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.max\_position\_embeddings - 1].
* **head\_mask** (numpy.ndarray of shape (batch\_size, sequence\_length), optional) -- Mask to nullify selected heads of the attention modules. Mask values selected in [0, 1]`:
  + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput) instead of a plain tuple.

**Returns**

[transformers.modeling\_flax\_outputs.FlaxQuestionAnsweringModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxQuestionAnsweringModelOutput)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_flax\_outputs.FlaxQuestionAnsweringModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_flax_outputs.FlaxQuestionAnsweringModelOutput) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([BertConfig](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertConfig)) and inputs.

* **start\_logits** (jnp.ndarray of shape (batch\_size, sequence\_length)) — Span-start scores (before SoftMax).
* **end\_logits** (jnp.ndarray of shape (batch\_size, sequence\_length)) — Span-end scores (before SoftMax).
* **hidden\_states** (tuple(jnp.ndarray), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of jnp.ndarray (one for the output of the embeddings + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the initial embedding outputs.

* **attentions** (tuple(jnp.ndarray), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of jnp.ndarray (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The FlaxBertPreTrainedModel forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, FlaxBertForQuestionAnswering

>>> tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

>>> model = FlaxBertForQuestionAnswering.from\_pretrained("bert-base-uncased")

>>> question, text = "Who was Jim Henson?", "Jim Henson was a nice puppet"

>>> inputs = tokenizer(question, text, return\_tensors="jax")

>>> outputs = model(\*\*inputs)

>>> start\_scores = outputs.start\_logits

>>> end\_scores = outputs.end\_logits