**Tokenizer**

A tokenizer is in charge of preparing the inputs for a model. The library contains tokenizers for all the models. Most of the tokenizers are available in two flavors: a full python implementation and a “Fast” implementation based on the Rust library [🤗 Tokenizers](https://github.com/huggingface/tokenizers). The “Fast” implementations allows:

1. a significant speed-up in particular when doing batched tokenization and
2. additional methods to map between the original string (character and words) and the token space (e.g. getting the index of the token comprising a given character or the span of characters corresponding to a given token).

The base classes [PreTrainedTokenizer](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizer) and [PreTrainedTokenizerFast](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast) implement the common methods for encoding string inputs in model inputs (see below) and instantiating/saving python and “Fast” tokenizers either from a local file or directory or from a pretrained tokenizer provided by the library (downloaded from HuggingFace’s AWS S3 repository). They both rely on [PreTrainedTokenizerBase](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.PreTrainedTokenizerBase) that contains the common methods, and [SpecialTokensMixin](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.SpecialTokensMixin).

[PreTrainedTokenizer](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizer) and [PreTrainedTokenizerFast](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast) thus implement the main methods for using all the tokenizers:

* Tokenizing (splitting strings in sub-word token strings), converting tokens strings to ids and back, and encoding/decoding (i.e., tokenizing and converting to integers).
* Adding new tokens to the vocabulary in a way that is independent of the underlying structure (BPE, SentencePiece…).
* Managing special tokens (like mask, beginning-of-sentence, etc.): adding them, assigning them to attributes in the tokenizer for easy access and making sure they are not split during tokenization.

[BatchEncoding](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.BatchEncoding) holds the output of the [PreTrainedTokenizerBase](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.PreTrainedTokenizerBase)’s encoding methods (\_\_call\_\_, encode\_plus and batch\_encode\_plus) and is derived from a Python dictionary. When the tokenizer is a pure python tokenizer, this class behaves just like a standard python dictionary and holds the various model inputs computed by these methods (input\_ids, attention\_mask…). When the tokenizer is a “Fast” tokenizer (i.e., backed by HuggingFace [tokenizers library](https://github.com/huggingface/tokenizers)), this class provides in addition several advanced alignment methods which can be used to map between the original string (character and words) and the token space (e.g., getting the index of the token comprising a given character or the span of characters corresponding to a given token).

**PreTrainedTokenizer**

classtransformers.**PreTrainedTokenizer**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils.py#L336)

( \*\*kwargs )

Expand 15 parameters

**Parameters**

* **model\_max\_length** (int, *optional*) — The maximum length (in number of tokens) for the inputs to the transformer model. When the tokenizer is loaded with [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.PreTrainedTokenizerBase.from_pretrained), this will be set to the value stored for the associated model in max\_model\_input\_sizes (see above). If no value is provided, will default to VERY\_LARGE\_INTEGER (int(1e30)).
* **padding\_side** (str, *optional*) — The side on which the model should have padding applied. Should be selected between [‘right’, ‘left’]. Default value is picked from the class attribute of the same name.
* **truncation\_side** (str, *optional*) — The side on which the model should have truncation applied. Should be selected between [‘right’, ‘left’]. Default value is picked from the class attribute of the same name.
* **chat\_template** (str, *optional*) — A Jinja template string that will be used to format lists of chat messages. See <https://huggingface.co/docs/transformers/chat_templating> for a full description.
* **model\_input\_names** (List[string], *optional*) — The list of inputs accepted by the forward pass of the model (like "token\_type\_ids" or "attention\_mask"). Default value is picked from the class attribute of the same name.
* **bos\_token** (str or tokenizers.AddedToken, *optional*) — A special token representing the beginning of a sentence. Will be associated to self.bos\_token and self.bos\_token\_id.
* **eos\_token** (str or tokenizers.AddedToken, *optional*) — A special token representing the end of a sentence. Will be associated to self.eos\_token and self.eos\_token\_id.
* **unk\_token** (str or tokenizers.AddedToken, *optional*) — A special token representing an out-of-vocabulary token. Will be associated to self.unk\_token and self.unk\_token\_id.
* **sep\_token** (str or tokenizers.AddedToken, *optional*) — A special token separating two different sentences in the same input (used by BERT for instance). Will be associated to self.sep\_token and self.sep\_token\_id.
* **pad\_token** (str or tokenizers.AddedToken, *optional*) — A special token used to make arrays of tokens the same size for batching purpose. Will then be ignored by attention mechanisms or loss computation. Will be associated to self.pad\_token and self.pad\_token\_id.
* **cls\_token** (str or tokenizers.AddedToken, *optional*) — A special token representing the class of the input (used by BERT for instance). Will be associated to self.cls\_token and self.cls\_token\_id.
* **mask\_token** (str or tokenizers.AddedToken, *optional*) — A special token representing a masked token (used by masked-language modeling pretraining objectives, like BERT). Will be associated to self.mask\_token and self.mask\_token\_id.
* **additional\_special\_tokens** (tuple or list of str or tokenizers.AddedToken, *optional*) — A tuple or a list of additional special tokens. Add them here to ensure they are skipped when decoding with skip\_special\_tokens is set to True. If they are not part of the vocabulary, they will be added at the end of the vocabulary.
* **clean\_up\_tokenization\_spaces** (bool, *optional*, defaults to True) — Whether or not the model should cleanup the spaces that were added when splitting the input text during the tokenization process.
* **split\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not the special tokens should be split during the tokenization process. The default behavior is to not split special tokens. This means that if <s> is the bos\_token, then tokenizer.tokenize("<s>") = ['<s>]. Otherwise, if split\_special\_tokens=True, then tokenizer.tokenize("<s>") will be give ['<', 's', '>']. This argument is only supported for slow tokenizers for the moment.

Base class for all slow tokenizers.

Inherits from [PreTrainedTokenizerBase](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.PreTrainedTokenizerBase).

Handle all the shared methods for tokenization and special tokens as well as methods downloading/caching/loading pretrained tokenizers as well as adding tokens to the vocabulary.

This class also contain the added tokens in a unified way on top of all tokenizers so we don’t have to handle the specific vocabulary augmentation methods of the various underlying dictionary structures (BPE, sentencepiece…).

Class attributes (overridden by derived classes)

* **vocab\_files\_names** (Dict[str, str]) — A dictionary with, as keys, the \_\_init\_\_ keyword name of each vocabulary file required by the model, and as associated values, the filename for saving the associated file (string).
* **pretrained\_vocab\_files\_map** (Dict[str, Dict[str, str]]) — A dictionary of dictionaries, with the high-level keys being the \_\_init\_\_ keyword name of each vocabulary file required by the model, the low-level being the short-cut-names of the pretrained models with, as associated values, the url to the associated pretrained vocabulary file.
* **max\_model\_input\_sizes** (Dict[str, Optional[int]]) — A dictionary with, as keys, the short-cut-names of the pretrained models, and as associated values, the maximum length of the sequence inputs of this model, or None if the model has no maximum input size.
* **pretrained\_init\_configuration** (Dict[str, Dict[str, Any]]) — A dictionary with, as keys, the short-cut-names of the pretrained models, and as associated values, a dictionary of specific arguments to pass to the \_\_init\_\_ method of the tokenizer class for this pretrained model when loading the tokenizer with the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.PreTrainedTokenizerBase.from_pretrained) method.
* **model\_input\_names** (List[str]) — A list of inputs expected in the forward pass of the model.
* **padding\_side** (str) — The default value for the side on which the model should have padding applied. Should be 'right' or 'left'.
* **truncation\_side** (str) — The default value for the side on which the model should have truncation applied. Should be 'right' or 'left'.

\_\_call\_\_

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L2728)

( text: typing.Union[str, typing.List[str], typing.List[typing.List[str]]] = Nonetext\_pair: typing.Union[str, typing.List[str], typing.List[typing.List[str]], NoneType] = Nonetext\_target: typing.Union[str, typing.List[str], typing.List[typing.List[str]]] = Nonetext\_pair\_target: typing.Union[str, typing.List[str], typing.List[typing.List[str]], NoneType] = Noneadd\_special\_tokens: bool = Truepadding: typing.Union[bool, str, transformers.utils.generic.PaddingStrategy] = Falsetruncation: typing.Union[bool, str, transformers.tokenization\_utils\_base.TruncationStrategy] = Nonemax\_length: typing.Optional[int] = Nonestride: int = 0is\_split\_into\_words: bool = Falsepad\_to\_multiple\_of: typing.Optional[int] = Nonereturn\_tensors: typing.Union[str, transformers.utils.generic.TensorType, NoneType] = Nonereturn\_token\_type\_ids: typing.Optional[bool] = Nonereturn\_attention\_mask: typing.Optional[bool] = Nonereturn\_overflowing\_tokens: bool = Falsereturn\_special\_tokens\_mask: bool = Falsereturn\_offsets\_mapping: bool = Falsereturn\_length: bool = Falseverbose: bool = True\*\*kwargs ) **→** [BatchEncoding](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.BatchEncoding)

Expand 19 parameters

**Parameters**

* **text** (str, List[str], List[List[str]], *optional*) — The sequence or batch of sequences to be encoded. Each sequence can be a string or a list of strings (pretokenized string). If the sequences are provided as list of strings (pretokenized), you must set is\_split\_into\_words=True (to lift the ambiguity with a batch of sequences).
* **text\_pair** (str, List[str], List[List[str]], *optional*) — The sequence or batch of sequences to be encoded. Each sequence can be a string or a list of strings (pretokenized string). If the sequences are provided as list of strings (pretokenized), you must set is\_split\_into\_words=True (to lift the ambiguity with a batch of sequences).
* **text\_target** (str, List[str], List[List[str]], *optional*) — The sequence or batch of sequences to be encoded as target texts. Each sequence can be a string or a list of strings (pretokenized string). If the sequences are provided as list of strings (pretokenized), you must set is\_split\_into\_words=True (to lift the ambiguity with a batch of sequences).
* **text\_pair\_target** (str, List[str], List[List[str]], *optional*) — The sequence or batch of sequences to be encoded as target texts. Each sequence can be a string or a list of strings (pretokenized string). If the sequences are provided as list of strings (pretokenized), you must set is\_split\_into\_words=True (to lift the ambiguity with a batch of sequences).
* **add\_special\_tokens** (bool, *optional*, defaults to True) — Whether or not to add special tokens when encoding the sequences. This will use the underlying PretrainedTokenizerBase.build\_inputs\_with\_special\_tokens function, which defines which tokens are automatically added to the input ids. This is usefull if you want to add bos or eos tokens automatically.
* **padding** (bool, str or [PaddingStrategy](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.utils.PaddingStrategy), *optional*, defaults to False) — Activates and controls padding. Accepts the following values:
  + True or 'longest': Pad to the longest sequence in the batch (or no padding if only a single sequence if provided).
  + 'max\_length': Pad to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided.
  + False or 'do\_not\_pad' (default): No padding (i.e., can output a batch with sequences of different lengths).
* **truncation** (bool, str or [TruncationStrategy](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.tokenization_utils_base.TruncationStrategy), *optional*, defaults to False) — Activates and controls truncation. Accepts the following values:
  + True or 'longest\_first': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will truncate token by token, removing a token from the longest sequence in the pair if a pair of sequences (or a batch of pairs) is provided.
  + 'only\_first': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will only truncate the first sequence of a pair if a pair of sequences (or a batch of pairs) is provided.
  + 'only\_second': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will only truncate the second sequence of a pair if a pair of sequences (or a batch of pairs) is provided.
  + False or 'do\_not\_truncate' (default): No truncation (i.e., can output batch with sequence lengths greater than the model maximum admissible input size).
* **max\_length** (int, *optional*) — Controls the maximum length to use by one of the truncation/padding parameters.

If left unset or set to None, this will use the predefined model maximum length if a maximum length is required by one of the truncation/padding parameters. If the model has no specific maximum input length (like XLNet) truncation/padding to a maximum length will be deactivated.

* **stride** (int, *optional*, defaults to 0) — If set to a number along with max\_length, the overflowing tokens returned when return\_overflowing\_tokens=True will contain some tokens from the end of the truncated sequence returned to provide some overlap between truncated and overflowing sequences. The value of this argument defines the number of overlapping tokens.
* **is\_split\_into\_words** (bool, *optional*, defaults to False) — Whether or not the input is already pre-tokenized (e.g., split into words). If set to True, the tokenizer assumes the input is already split into words (for instance, by splitting it on whitespace) which it will tokenize. This is useful for NER or token classification.
* **pad\_to\_multiple\_of** (int, *optional*) — If set will pad the sequence to a multiple of the provided value. Requires padding to be activated. This is especially useful to enable the use of Tensor Cores on NVIDIA hardware with compute capability >= 7.5 (Volta).
* **return\_tensors** (str or [TensorType](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.TensorType), *optional*) — If set, will return tensors instead of list of python integers. Acceptable values are:
  + 'tf': Return TensorFlow tf.constant objects.
  + 'pt': Return PyTorch torch.Tensor objects.
  + 'np': Return Numpy np.ndarray objects.
* **return\_token\_type\_ids** (bool, *optional*) — Whether to return token type IDs. If left to the default, will return the token type IDs according to the specific tokenizer’s default, defined by the return\_outputs attribute.

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

* **return\_attention\_mask** (bool, *optional*) — Whether to return the attention mask. If left to the default, will return the attention mask according to the specific tokenizer’s default, defined by the return\_outputs attribute.

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

* **return\_overflowing\_tokens** (bool, *optional*, defaults to False) — Whether or not to return overflowing token sequences. If a pair of sequences of input ids (or a batch of pairs) is provided with truncation\_strategy = longest\_first or True, an error is raised instead of returning overflowing tokens.
* **return\_special\_tokens\_mask** (bool, *optional*, defaults to False) — Whether or not to return special tokens mask information.
* **return\_offsets\_mapping** (bool, *optional*, defaults to False) — Whether or not to return (char\_start, char\_end) for each token.

This is only available on fast tokenizers inheriting from [PreTrainedTokenizerFast](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast), if using Python’s tokenizer, this method will raise NotImplementedError.

* **return\_length** (bool, *optional*, defaults to False) — Whether or not to return the lengths of the encoded inputs.
* **verbose** (bool, *optional*, defaults to True) — Whether or not to print more information and warnings. \*\*kwargs — passed to the self.tokenize() method

**Returns**

[BatchEncoding](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.BatchEncoding)

A [BatchEncoding](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.BatchEncoding) with the following fields:

* **input\_ids** — List of token ids to be fed to a model.

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

* **token\_type\_ids** — List of token type ids to be fed to a model (when return\_token\_type\_ids=True or if *“token\_type\_ids”* is in self.model\_input\_names).

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

* **attention\_mask** — List of indices specifying which tokens should be attended to by the model (when return\_attention\_mask=True or if *“attention\_mask”* is in self.model\_input\_names).

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

* **overflowing\_tokens** — List of overflowing tokens sequences (when a max\_length is specified and return\_overflowing\_tokens=True).
* **num\_truncated\_tokens** — Number of tokens truncated (when a max\_length is specified and return\_overflowing\_tokens=True).
* **special\_tokens\_mask** — List of 0s and 1s, with 1 specifying added special tokens and 0 specifying regular sequence tokens (when add\_special\_tokens=True and return\_special\_tokens\_mask=True).
* **length** — The length of the inputs (when return\_length=True)

Main method to tokenize and prepare for the model one or several sequence(s) or one or several pair(s) of sequences.

add\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L974)

( new\_tokens: typing.Union[str, tokenizers.AddedToken, typing.List[typing.Union[str, tokenizers.AddedToken]]]special\_tokens: bool = False ) **→** int

**Parameters**

* **new\_tokens** (str, tokenizers.AddedToken or a list of *str* or tokenizers.AddedToken) — Tokens are only added if they are not already in the vocabulary. tokenizers.AddedToken wraps a string token to let you personalize its behavior: whether this token should only match against a single word, whether this token should strip all potential whitespaces on the left side, whether this token should strip all potential whitespaces on the right side, etc.
* **special\_tokens** (bool, *optional*, defaults to False) — Can be used to specify if the token is a special token. This mostly change the normalization behavior (special tokens like CLS or [MASK] are usually not lower-cased for instance).

See details for tokenizers.AddedToken in HuggingFace tokenizers library.

**Returns**

int

Number of tokens added to the vocabulary.

Add a list of new tokens to the tokenizer class. If the new tokens are not in the vocabulary, they are added to it with indices starting from length of the current vocabulary and and will be isolated before the tokenization algorithm is applied. Added tokens and tokens from the vocabulary of the tokenization algorithm are therefore not treated in the same way.

Note, when adding new tokens to the vocabulary, you should make sure to also resize the token embedding matrix of the model so that its embedding matrix matches the tokenizer.

In order to do that, please use the [resize\_token\_embeddings()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.resize_token_embeddings) method.

Examples:

Copied

*# Let's see how to increase the vocabulary of Bert model and tokenizer*

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

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

num\_added\_toks = tokenizer.add\_tokens(["new\_tok1", "my\_new-tok2"])

print("We have added", num\_added\_toks, "tokens")

*# Notice: resize\_token\_embeddings expect to receive the full size of the new vocabulary, i.e., the length of the tokenizer.*

model.resize\_token\_embeddings(len(tokenizer))

add\_special\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L873)

( special\_tokens\_dict: typing.Dict[str, typing.Union[str, tokenizers.AddedToken]]replace\_additional\_special\_tokens = True ) **→** int

**Parameters**

* **special\_tokens\_dict** (dictionary *str* to *str* or tokenizers.AddedToken) — Keys should be in the list of predefined special attributes: [bos\_token, eos\_token, unk\_token, sep\_token, pad\_token, cls\_token, mask\_token, additional\_special\_tokens].

Tokens are only added if they are not already in the vocabulary (tested by checking if the tokenizer assign the index of the unk\_token to them).

* **replace\_additional\_special\_tokens** (bool, *optional*,, defaults to True) — If True, the existing list of additional special tokens will be replaced by the list provided in special\_tokens\_dict. Otherwise, self.\_additional\_special\_tokens is just extended. In the former case, the tokens will NOT be removed from the tokenizer’s full vocabulary - they are only being flagged as non-special tokens. Remember, this only affects which tokens are skipped during decoding, not the added\_tokens\_encoder and added\_tokens\_decoder. This means that the previous additional\_special\_tokens are still added tokens, and will not be split by the model.

**Returns**

int

Number of tokens added to the vocabulary.

Add a dictionary of special tokens (eos, pad, cls, etc.) to the encoder and link them to class attributes. If special tokens are NOT in the vocabulary, they are added to it (indexed starting from the last index of the current vocabulary).

When adding new tokens to the vocabulary, you should make sure to also resize the token embedding matrix of the model so that its embedding matrix matches the tokenizer.

In order to do that, please use the [resize\_token\_embeddings()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.resize_token_embeddings) method.

Using add\_special\_tokens will ensure your special tokens can be used in several ways:

* Special tokens can be skipped when decoding using skip\_special\_tokens = True.
* Special tokens are carefully handled by the tokenizer (they are never split), similar to AddedTokens.
* You can easily refer to special tokens using tokenizer class attributes like tokenizer.cls\_token. This makes it easy to develop model-agnostic training and fine-tuning scripts.

When possible, special tokens are already registered for provided pretrained models (for instance [BertTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertTokenizer) cls\_token is already registered to be :obj*’[CLS]’* and XLM’s one is also registered to be '</s>').

Examples:

Copied

*# Let's see how to add a new classification token to GPT-2*

tokenizer = GPT2Tokenizer.from\_pretrained("gpt2")

model = GPT2Model.from\_pretrained("gpt2")

special\_tokens\_dict = {"cls\_token": "<CLS>"}

num\_added\_toks = tokenizer.add\_special\_tokens(special\_tokens\_dict)

print("We have added", num\_added\_toks, "tokens")

*# Notice: resize\_token\_embeddings expect to receive the full size of the new vocabulary, i.e., the length of the tokenizer.*

model.resize\_token\_embeddings(len(tokenizer))

assert tokenizer.cls\_token == "<CLS>"

apply\_chat\_template

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L1677)

( conversation: typing.Union[typing.List[typing.Dict[str, str]], ForwardRef('Conversation')]chat\_template: typing.Optional[str] = Noneadd\_generation\_prompt: bool = Falsetokenize: bool = Truepadding: bool = Falsetruncation: bool = Falsemax\_length: typing.Optional[int] = Nonereturn\_tensors: typing.Union[str, transformers.utils.generic.TensorType, NoneType] = None\*\*tokenizer\_kwargs ) **→** List[int]

Expand 8 parameters

**Parameters**

* **conversation** (Union[List[Dict[str, str]], “Conversation”]) — A Conversation object or list of dicts with “role” and “content” keys, representing the chat history so far.
* **chat\_template** (str, *optional*) — A Jinja template to use for this conversion. If this is not passed, the model’s default chat template will be used instead.
* **add\_generation\_prompt** (bool, *optional*) — Whether to end the prompt with the token(s) that indicate the start of an assistant message. This is useful when you want to generate a response from the model. Note that this argument will be passed to the chat template, and so it must be supported in the template for this argument to have any effect.
* **tokenize** (bool, defaults to True) — Whether to tokenize the output. If False, the output will be a string.
* **padding** (bool, defaults to False) — Whether to pad sequences to the maximum length. Has no effect if tokenize is False.
* **truncation** (bool, defaults to False) — Whether to truncate sequences at the maximum length. Has no effect if tokenize is False.
* **max\_length** (int, *optional*) — Maximum length (in tokens) to use for padding or truncation. Has no effect if tokenize is False. If not specified, the tokenizer’s max\_length attribute will be used as a default.
* **return\_tensors** (str or [TensorType](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.TensorType), *optional*) — If set, will return tensors of a particular framework. Has no effect if tokenize is False. Acceptable values are:
  + 'tf': Return TensorFlow tf.Tensor objects.
  + 'pt': Return PyTorch torch.Tensor objects.
  + 'np': Return NumPy np.ndarray objects.
  + 'jax': Return JAX jnp.ndarray objects. \*\*tokenizer\_kwargs — Additional kwargs to pass to the tokenizer.

**Returns**

List[int]

A list of token ids representing the tokenized chat so far, including control tokens. This output is ready to pass to the model, either directly or via methods like generate().

Converts a Conversation object or a list of dictionaries with "role" and "content" keys to a list of token ids. This method is intended for use with chat models, and will read the tokenizer’s chat\_template attribute to determine the format and control tokens to use when converting. When chat\_template is None, it will fall back to the default\_chat\_template specified at the class level.

batch\_decode

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L3686)

( sequences: typing.Union[typing.List[int], typing.List[typing.List[int]], ForwardRef('np.ndarray'), ForwardRef('torch.Tensor'), ForwardRef('tf.Tensor')]skip\_special\_tokens: bool = Falseclean\_up\_tokenization\_spaces: bool = None\*\*kwargs ) **→** List[str]

**Parameters**

* **sequences** (Union[List[int], List[List[int]], np.ndarray, torch.Tensor, tf.Tensor]) — List of tokenized input ids. Can be obtained using the \_\_call\_\_ method.
* **skip\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not to remove special tokens in the decoding.
* **clean\_up\_tokenization\_spaces** (bool, *optional*) — Whether or not to clean up the tokenization spaces. If None, will default to self.clean\_up\_tokenization\_spaces.
* **kwargs** (additional keyword arguments, *optional*) — Will be passed to the underlying model specific decode method.

**Returns**

List[str]

The list of decoded sentences.

Convert a list of lists of token ids into a list of strings by calling decode.

decode

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L3720)

( token\_ids: typing.Union[int, typing.List[int], ForwardRef('np.ndarray'), ForwardRef('torch.Tensor'), ForwardRef('tf.Tensor')]skip\_special\_tokens: bool = Falseclean\_up\_tokenization\_spaces: bool = None\*\*kwargs ) **→** str

**Parameters**

* **token\_ids** (Union[int, List[int], np.ndarray, torch.Tensor, tf.Tensor]) — List of tokenized input ids. Can be obtained using the \_\_call\_\_ method.
* **skip\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not to remove special tokens in the decoding.
* **clean\_up\_tokenization\_spaces** (bool, *optional*) — Whether or not to clean up the tokenization spaces. If None, will default to self.clean\_up\_tokenization\_spaces.
* **kwargs** (additional keyword arguments, *optional*) — Will be passed to the underlying model specific decode method.

**Returns**

str

The decoded sentence.

Converts a sequence of ids in a string, using the tokenizer and vocabulary with options to remove special tokens and clean up tokenization spaces.

Similar to doing self.convert\_tokens\_to\_string(self.convert\_ids\_to\_tokens(token\_ids)).

encode

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L2536)

( text: typing.Union[str, typing.List[str], typing.List[int]]text\_pair: typing.Union[str, typing.List[str], typing.List[int], NoneType] = Noneadd\_special\_tokens: bool = Truepadding: typing.Union[bool, str, transformers.utils.generic.PaddingStrategy] = Falsetruncation: typing.Union[bool, str, transformers.tokenization\_utils\_base.TruncationStrategy] = Nonemax\_length: typing.Optional[int] = Nonestride: int = 0return\_tensors: typing.Union[str, transformers.utils.generic.TensorType, NoneType] = None\*\*kwargs ) **→** List[int], torch.Tensor, tf.Tensor or np.ndarray

Expand 10 parameters

**Parameters**

* **text** (str, List[str] or List[int]) — The first sequence to be encoded. This can be a string, a list of strings (tokenized string using the tokenize method) or a list of integers (tokenized string ids using the convert\_tokens\_to\_ids method).
* **text\_pair** (str, List[str] or List[int], *optional*) — Optional second sequence to be encoded. This can be a string, a list of strings (tokenized string using the tokenize method) or a list of integers (tokenized string ids using the convert\_tokens\_to\_ids method).
* **add\_special\_tokens** (bool, *optional*, defaults to True) — Whether or not to add special tokens when encoding the sequences. This will use the underlying PretrainedTokenizerBase.build\_inputs\_with\_special\_tokens function, which defines which tokens are automatically added to the input ids. This is usefull if you want to add bos or eos tokens automatically.
* **padding** (bool, str or [PaddingStrategy](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.utils.PaddingStrategy), *optional*, defaults to False) — Activates and controls padding. Accepts the following values:
  + True or 'longest': Pad to the longest sequence in the batch (or no padding if only a single sequence if provided).
  + 'max\_length': Pad to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided.
  + False or 'do\_not\_pad' (default): No padding (i.e., can output a batch with sequences of different lengths).
* **truncation** (bool, str or [TruncationStrategy](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.tokenization_utils_base.TruncationStrategy), *optional*, defaults to False) — Activates and controls truncation. Accepts the following values:
  + True or 'longest\_first': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will truncate token by token, removing a token from the longest sequence in the pair if a pair of sequences (or a batch of pairs) is provided.
  + 'only\_first': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will only truncate the first sequence of a pair if a pair of sequences (or a batch of pairs) is provided.
  + 'only\_second': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will only truncate the second sequence of a pair if a pair of sequences (or a batch of pairs) is provided.
  + False or 'do\_not\_truncate' (default): No truncation (i.e., can output batch with sequence lengths greater than the model maximum admissible input size).
* **max\_length** (int, *optional*) — Controls the maximum length to use by one of the truncation/padding parameters.

If left unset or set to None, this will use the predefined model maximum length if a maximum length is required by one of the truncation/padding parameters. If the model has no specific maximum input length (like XLNet) truncation/padding to a maximum length will be deactivated.

* **stride** (int, *optional*, defaults to 0) — If set to a number along with max\_length, the overflowing tokens returned when return\_overflowing\_tokens=True will contain some tokens from the end of the truncated sequence returned to provide some overlap between truncated and overflowing sequences. The value of this argument defines the number of overlapping tokens.
* **is\_split\_into\_words** (bool, *optional*, defaults to False) — Whether or not the input is already pre-tokenized (e.g., split into words). If set to True, the tokenizer assumes the input is already split into words (for instance, by splitting it on whitespace) which it will tokenize. This is useful for NER or token classification.
* **pad\_to\_multiple\_of** (int, *optional*) — If set will pad the sequence to a multiple of the provided value. Requires padding to be activated. This is especially useful to enable the use of Tensor Cores on NVIDIA hardware with compute capability >= 7.5 (Volta).
* **return\_tensors** (str or [TensorType](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.TensorType), *optional*) — If set, will return tensors instead of list of python integers. Acceptable values are:
  + 'tf': Return TensorFlow tf.constant objects.
  + 'pt': Return PyTorch torch.Tensor objects.
  + 'np': Return Numpy np.ndarray objects.

\*\*kwargs — Passed along to the .tokenize() method.

**Returns**

List[int]**,**torch.Tensor**,**tf.Tensor**or**np.ndarray

The tokenized ids of the text.

Converts a string to a sequence of ids (integer), using the tokenizer and vocabulary.

Same as doing self.convert\_tokens\_to\_ids(self.tokenize(text)).

push\_to\_hub

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/utils/hub.py#L753)

( repo\_id: struse\_temp\_dir: typing.Optional[bool] = Nonecommit\_message: typing.Optional[str] = Noneprivate: typing.Optional[bool] = Nonetoken: typing.Union[bool, str, NoneType] = Nonemax\_shard\_size: typing.Union[int, str, NoneType] = '5GB'create\_pr: bool = Falsesafe\_serialization: bool = Truerevision: str = Nonecommit\_description: str = None\*\*deprecated\_kwargs )

Expand 10 parameters

**Parameters**

* **repo\_id** (str) — The name of the repository you want to push your tokenizer to. It should contain your organization name when pushing to a given organization.
* **use\_temp\_dir** (bool, *optional*) — Whether or not to use a temporary directory to store the files saved before they are pushed to the Hub. Will default to True if there is no directory named like repo\_id, False otherwise.
* **commit\_message** (str, *optional*) — Message to commit while pushing. Will default to "Upload tokenizer".
* **private** (bool, *optional*) — Whether or not the repository created should be private.
* **token** (bool or str, *optional*) — The token to use as HTTP bearer authorization for remote files. If True, will use the token generated when running huggingface-cli login (stored in ~/.huggingface). Will default to True if repo\_url is not specified.
* **max\_shard\_size** (int or str, *optional*, defaults to "5GB") — Only applicable for models. The maximum size for a checkpoint before being sharded. Checkpoints shard will then be each of size lower than this size. If expressed as a string, needs to be digits followed by a unit (like "5MB"). We default it to "5GB" so that users can easily load models on free-tier Google Colab instances without any CPU OOM issues.
* **create\_pr** (bool, *optional*, defaults to False) — Whether or not to create a PR with the uploaded files or directly commit.
* **safe\_serialization** (bool, *optional*, defaults to True) — Whether or not to convert the model weights in safetensors format for safer serialization.
* **revision** (str, *optional*) — Branch to push the uploaded files to.
* **commit\_description** (str, *optional*) — The description of the commit that will be created

Upload the tokenizer files to the 🤗 Model Hub.

Examples:

Copied

from transformers import AutoTokenizer

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

*# Push the tokenizer to your namespace with the name "my-finetuned-bert".*

tokenizer.push\_to\_hub("my-finetuned-bert")

*# Push the tokenizer to an organization with the name "my-finetuned-bert".*

tokenizer.push\_to\_hub("huggingface/my-finetuned-bert")

convert\_ids\_to\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils.py#L953)

( ids: typing.Union[int, typing.List[int]]skip\_special\_tokens: bool = False ) **→** str or List[str]

**Parameters**

* **ids** (int or List[int]) — The token id (or token ids) to convert to tokens.
* **skip\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not to remove special tokens in the decoding.

**Returns**

str**or**List[str]

The decoded token(s).

Converts a single index or a sequence of indices in a token or a sequence of tokens, using the vocabulary and added tokens.

convert\_tokens\_to\_ids

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils.py#L630)

( tokens: typing.Union[str, typing.List[str]] ) **→** int or List[int]

**Parameters**

* **tokens** (str or List[str]) — One or several token(s) to convert to token id(s).

**Returns**

int**or**List[int]

The token id or list of token ids.

Converts a token string (or a sequence of tokens) in a single integer id (or a sequence of ids), using the vocabulary.

get\_added\_vocab

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils.py#L415)

( ) **→** Dict[str, int]

**Returns**

Dict[str, int]

The added tokens.

Returns the added tokens in the vocabulary as a dictionary of token to index. Results might be different from the fast call because for now we always add the tokens even if they are already in the vocabulary. This is something we should change.

num\_special\_tokens\_to\_add

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils.py#L518)

( pair: bool = False ) **→** int

**Parameters**

* **pair** (bool, *optional*, defaults to False) — Whether the number of added tokens should be computed in the case of a sequence pair or a single sequence.

**Returns**

int

Number of special tokens added to sequences.

Returns the number of added tokens when encoding a sequence with special tokens.

This encodes a dummy input and checks the number of added tokens, and is therefore not efficient. Do not put this inside your training loop.

prepare\_for\_tokenization

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils.py#L891)

( text: stris\_split\_into\_words: bool = False\*\*kwargs ) **→** Tuple[str, Dict[str, Any]]

**Parameters**

* **text** (str) — The text to prepare.
* **is\_split\_into\_words** (bool, *optional*, defaults to False) — Whether or not the input is already pre-tokenized (e.g., split into words). If set to True, the tokenizer assumes the input is already split into words (for instance, by splitting it on whitespace) which it will tokenize. This is useful for NER or token classification.
* **kwargs** (Dict[str, Any], *optional*) — Keyword arguments to use for the tokenization.

**Returns**

Tuple[str, Dict[str, Any]]

The prepared text and the unused kwargs.

Performs any necessary transformations before tokenization.

This method should pop the arguments from kwargs and return the remaining kwargs as well. We test the kwargs at the end of the encoding process to be sure all the arguments have been used.

tokenize

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils.py#L541)

( text: str\*\*kwargs ) **→** List[str]

**Parameters**

* **text** (str) — The sequence to be encoded.
* \***\*kwargs** (additional keyword arguments) — Passed along to the model-specific prepare\_for\_tokenization preprocessing method.

**Returns**

List[str]

The list of tokens.

Converts a string into a sequence of tokens, using the tokenizer.

Split in words for word-based vocabulary or sub-words for sub-word-based vocabularies (BPE/SentencePieces/WordPieces). Takes care of added tokens.

**PreTrainedTokenizerFast**

The [PreTrainedTokenizerFast](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast) depend on the [tokenizers](https://huggingface.co/docs/tokenizers) library. The tokenizers obtained from the 🤗 tokenizers library can be loaded very simply into 🤗 transformers. Take a look at the [Using tokenizers from 🤗 tokenizers](https://huggingface.co/docs/transformers/main/fast_tokenizers) page to understand how this is done.

classtransformers.**PreTrainedTokenizerFast**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_fast.py#L78)

( \*args\*\*kwargs )

Expand 17 parameters

**Parameters**

* **model\_max\_length** (int, *optional*) — The maximum length (in number of tokens) for the inputs to the transformer model. When the tokenizer is loaded with [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.PreTrainedTokenizerBase.from_pretrained), this will be set to the value stored for the associated model in max\_model\_input\_sizes (see above). If no value is provided, will default to VERY\_LARGE\_INTEGER (int(1e30)).
* **padding\_side** (str, *optional*) — The side on which the model should have padding applied. Should be selected between [‘right’, ‘left’]. Default value is picked from the class attribute of the same name.
* **truncation\_side** (str, *optional*) — The side on which the model should have truncation applied. Should be selected between [‘right’, ‘left’]. Default value is picked from the class attribute of the same name.
* **chat\_template** (str, *optional*) — A Jinja template string that will be used to format lists of chat messages. See <https://huggingface.co/docs/transformers/chat_templating> for a full description.
* **model\_input\_names** (List[string], *optional*) — The list of inputs accepted by the forward pass of the model (like "token\_type\_ids" or "attention\_mask"). Default value is picked from the class attribute of the same name.
* **bos\_token** (str or tokenizers.AddedToken, *optional*) — A special token representing the beginning of a sentence. Will be associated to self.bos\_token and self.bos\_token\_id.
* **eos\_token** (str or tokenizers.AddedToken, *optional*) — A special token representing the end of a sentence. Will be associated to self.eos\_token and self.eos\_token\_id.
* **unk\_token** (str or tokenizers.AddedToken, *optional*) — A special token representing an out-of-vocabulary token. Will be associated to self.unk\_token and self.unk\_token\_id.
* **sep\_token** (str or tokenizers.AddedToken, *optional*) — A special token separating two different sentences in the same input (used by BERT for instance). Will be associated to self.sep\_token and self.sep\_token\_id.
* **pad\_token** (str or tokenizers.AddedToken, *optional*) — A special token used to make arrays of tokens the same size for batching purpose. Will then be ignored by attention mechanisms or loss computation. Will be associated to self.pad\_token and self.pad\_token\_id.
* **cls\_token** (str or tokenizers.AddedToken, *optional*) — A special token representing the class of the input (used by BERT for instance). Will be associated to self.cls\_token and self.cls\_token\_id.
* **mask\_token** (str or tokenizers.AddedToken, *optional*) — A special token representing a masked token (used by masked-language modeling pretraining objectives, like BERT). Will be associated to self.mask\_token and self.mask\_token\_id.
* **additional\_special\_tokens** (tuple or list of str or tokenizers.AddedToken, *optional*) — A tuple or a list of additional special tokens. Add them here to ensure they are skipped when decoding with skip\_special\_tokens is set to True. If they are not part of the vocabulary, they will be added at the end of the vocabulary.
* **clean\_up\_tokenization\_spaces** (bool, *optional*, defaults to True) — Whether or not the model should cleanup the spaces that were added when splitting the input text during the tokenization process.
* **split\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not the special tokens should be split during the tokenization process. The default behavior is to not split special tokens. This means that if <s> is the bos\_token, then tokenizer.tokenize("<s>") = ['<s>]. Otherwise, if split\_special\_tokens=True, then tokenizer.tokenize("<s>") will be give ['<', 's', '>']. This argument is only supported for slow tokenizers for the moment.
* **tokenizer\_object** ([tokenizers.Tokenizer](https://huggingface.co/docs/tokenizers/main/en/api/tokenizer#tokenizers.Tokenizer)) — A [tokenizers.Tokenizer](https://huggingface.co/docs/tokenizers/main/en/api/tokenizer#tokenizers.Tokenizer) object from 🤗 tokenizers to instantiate from. See [Using tokenizers from 🤗 tokenizers](https://huggingface.co/docs/transformers/main/fast_tokenizers) for more information.
* **tokenizer\_file** (str) — A path to a local JSON file representing a previously serialized [tokenizers.Tokenizer](https://huggingface.co/docs/tokenizers/main/en/api/tokenizer#tokenizers.Tokenizer) object from 🤗 tokenizers.

Base class for all fast tokenizers (wrapping HuggingFace tokenizers library).

Inherits from [PreTrainedTokenizerBase](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.PreTrainedTokenizerBase).

Handles all the shared methods for tokenization and special tokens, as well as methods for downloading/caching/loading pretrained tokenizers, as well as adding tokens to the vocabulary.

This class also contains the added tokens in a unified way on top of all tokenizers so we don’t have to handle the specific vocabulary augmentation methods of the various underlying dictionary structures (BPE, sentencepiece…).

Class attributes (overridden by derived classes)

* **vocab\_files\_names** (Dict[str, str]) — A dictionary with, as keys, the \_\_init\_\_ keyword name of each vocabulary file required by the model, and as associated values, the filename for saving the associated file (string).
* **pretrained\_vocab\_files\_map** (Dict[str, Dict[str, str]]) — A dictionary of dictionaries, with the high-level keys being the \_\_init\_\_ keyword name of each vocabulary file required by the model, the low-level being the short-cut-names of the pretrained models with, as associated values, the url to the associated pretrained vocabulary file.
* **max\_model\_input\_sizes** (Dict[str, Optional[int]]) — A dictionary with, as keys, the short-cut-names of the pretrained models, and as associated values, the maximum length of the sequence inputs of this model, or None if the model has no maximum input size.
* **pretrained\_init\_configuration** (Dict[str, Dict[str, Any]]) — A dictionary with, as keys, the short-cut-names of the pretrained models, and as associated values, a dictionary of specific arguments to pass to the \_\_init\_\_ method of the tokenizer class for this pretrained model when loading the tokenizer with the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.PreTrainedTokenizerBase.from_pretrained) method.
* **model\_input\_names** (List[str]) — A list of inputs expected in the forward pass of the model.
* **padding\_side** (str) — The default value for the side on which the model should have padding applied. Should be 'right' or 'left'.
* **truncation\_side** (str) — The default value for the side on which the model should have truncation applied. Should be 'right' or 'left'.

\_\_call\_\_

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L2728)

( text: typing.Union[str, typing.List[str], typing.List[typing.List[str]]] = Nonetext\_pair: typing.Union[str, typing.List[str], typing.List[typing.List[str]], NoneType] = Nonetext\_target: typing.Union[str, typing.List[str], typing.List[typing.List[str]]] = Nonetext\_pair\_target: typing.Union[str, typing.List[str], typing.List[typing.List[str]], NoneType] = Noneadd\_special\_tokens: bool = Truepadding: typing.Union[bool, str, transformers.utils.generic.PaddingStrategy] = Falsetruncation: typing.Union[bool, str, transformers.tokenization\_utils\_base.TruncationStrategy] = Nonemax\_length: typing.Optional[int] = Nonestride: int = 0is\_split\_into\_words: bool = Falsepad\_to\_multiple\_of: typing.Optional[int] = Nonereturn\_tensors: typing.Union[str, transformers.utils.generic.TensorType, NoneType] = Nonereturn\_token\_type\_ids: typing.Optional[bool] = Nonereturn\_attention\_mask: typing.Optional[bool] = Nonereturn\_overflowing\_tokens: bool = Falsereturn\_special\_tokens\_mask: bool = Falsereturn\_offsets\_mapping: bool = Falsereturn\_length: bool = Falseverbose: bool = True\*\*kwargs ) **→** [BatchEncoding](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.BatchEncoding)

Expand 19 parameters

**Parameters**

* **text** (str, List[str], List[List[str]], *optional*) — The sequence or batch of sequences to be encoded. Each sequence can be a string or a list of strings (pretokenized string). If the sequences are provided as list of strings (pretokenized), you must set is\_split\_into\_words=True (to lift the ambiguity with a batch of sequences).
* **text\_pair** (str, List[str], List[List[str]], *optional*) — The sequence or batch of sequences to be encoded. Each sequence can be a string or a list of strings (pretokenized string). If the sequences are provided as list of strings (pretokenized), you must set is\_split\_into\_words=True (to lift the ambiguity with a batch of sequences).
* **text\_target** (str, List[str], List[List[str]], *optional*) — The sequence or batch of sequences to be encoded as target texts. Each sequence can be a string or a list of strings (pretokenized string). If the sequences are provided as list of strings (pretokenized), you must set is\_split\_into\_words=True (to lift the ambiguity with a batch of sequences).
* **text\_pair\_target** (str, List[str], List[List[str]], *optional*) — The sequence or batch of sequences to be encoded as target texts. Each sequence can be a string or a list of strings (pretokenized string). If the sequences are provided as list of strings (pretokenized), you must set is\_split\_into\_words=True (to lift the ambiguity with a batch of sequences).
* **add\_special\_tokens** (bool, *optional*, defaults to True) — Whether or not to add special tokens when encoding the sequences. This will use the underlying PretrainedTokenizerBase.build\_inputs\_with\_special\_tokens function, which defines which tokens are automatically added to the input ids. This is usefull if you want to add bos or eos tokens automatically.
* **padding** (bool, str or [PaddingStrategy](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.utils.PaddingStrategy), *optional*, defaults to False) — Activates and controls padding. Accepts the following values:
  + True or 'longest': Pad to the longest sequence in the batch (or no padding if only a single sequence if provided).
  + 'max\_length': Pad to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided.
  + False or 'do\_not\_pad' (default): No padding (i.e., can output a batch with sequences of different lengths).
* **truncation** (bool, str or [TruncationStrategy](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.tokenization_utils_base.TruncationStrategy), *optional*, defaults to False) — Activates and controls truncation. Accepts the following values:
  + True or 'longest\_first': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will truncate token by token, removing a token from the longest sequence in the pair if a pair of sequences (or a batch of pairs) is provided.
  + 'only\_first': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will only truncate the first sequence of a pair if a pair of sequences (or a batch of pairs) is provided.
  + 'only\_second': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will only truncate the second sequence of a pair if a pair of sequences (or a batch of pairs) is provided.
  + False or 'do\_not\_truncate' (default): No truncation (i.e., can output batch with sequence lengths greater than the model maximum admissible input size).
* **max\_length** (int, *optional*) — Controls the maximum length to use by one of the truncation/padding parameters.

If left unset or set to None, this will use the predefined model maximum length if a maximum length is required by one of the truncation/padding parameters. If the model has no specific maximum input length (like XLNet) truncation/padding to a maximum length will be deactivated.

* **stride** (int, *optional*, defaults to 0) — If set to a number along with max\_length, the overflowing tokens returned when return\_overflowing\_tokens=True will contain some tokens from the end of the truncated sequence returned to provide some overlap between truncated and overflowing sequences. The value of this argument defines the number of overlapping tokens.
* **is\_split\_into\_words** (bool, *optional*, defaults to False) — Whether or not the input is already pre-tokenized (e.g., split into words). If set to True, the tokenizer assumes the input is already split into words (for instance, by splitting it on whitespace) which it will tokenize. This is useful for NER or token classification.
* **pad\_to\_multiple\_of** (int, *optional*) — If set will pad the sequence to a multiple of the provided value. Requires padding to be activated. This is especially useful to enable the use of Tensor Cores on NVIDIA hardware with compute capability >= 7.5 (Volta).
* **return\_tensors** (str or [TensorType](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.TensorType), *optional*) — If set, will return tensors instead of list of python integers. Acceptable values are:
  + 'tf': Return TensorFlow tf.constant objects.
  + 'pt': Return PyTorch torch.Tensor objects.
  + 'np': Return Numpy np.ndarray objects.
* **return\_token\_type\_ids** (bool, *optional*) — Whether to return token type IDs. If left to the default, will return the token type IDs according to the specific tokenizer’s default, defined by the return\_outputs attribute.

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

* **return\_attention\_mask** (bool, *optional*) — Whether to return the attention mask. If left to the default, will return the attention mask according to the specific tokenizer’s default, defined by the return\_outputs attribute.

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

* **return\_overflowing\_tokens** (bool, *optional*, defaults to False) — Whether or not to return overflowing token sequences. If a pair of sequences of input ids (or a batch of pairs) is provided with truncation\_strategy = longest\_first or True, an error is raised instead of returning overflowing tokens.
* **return\_special\_tokens\_mask** (bool, *optional*, defaults to False) — Whether or not to return special tokens mask information.
* **return\_offsets\_mapping** (bool, *optional*, defaults to False) — Whether or not to return (char\_start, char\_end) for each token.

This is only available on fast tokenizers inheriting from [PreTrainedTokenizerFast](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast), if using Python’s tokenizer, this method will raise NotImplementedError.

* **return\_length** (bool, *optional*, defaults to False) — Whether or not to return the lengths of the encoded inputs.
* **verbose** (bool, *optional*, defaults to True) — Whether or not to print more information and warnings. \*\*kwargs — passed to the self.tokenize() method

**Returns**

[BatchEncoding](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.BatchEncoding)

A [BatchEncoding](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.BatchEncoding) with the following fields:

* **input\_ids** — List of token ids to be fed to a model.

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

* **token\_type\_ids** — List of token type ids to be fed to a model (when return\_token\_type\_ids=True or if *“token\_type\_ids”* is in self.model\_input\_names).

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

* **attention\_mask** — List of indices specifying which tokens should be attended to by the model (when return\_attention\_mask=True or if *“attention\_mask”* is in self.model\_input\_names).

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

* **overflowing\_tokens** — List of overflowing tokens sequences (when a max\_length is specified and return\_overflowing\_tokens=True).
* **num\_truncated\_tokens** — Number of tokens truncated (when a max\_length is specified and return\_overflowing\_tokens=True).
* **special\_tokens\_mask** — List of 0s and 1s, with 1 specifying added special tokens and 0 specifying regular sequence tokens (when add\_special\_tokens=True and return\_special\_tokens\_mask=True).
* **length** — The length of the inputs (when return\_length=True)

Main method to tokenize and prepare for the model one or several sequence(s) or one or several pair(s) of sequences.

add\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L974)

( new\_tokens: typing.Union[str, tokenizers.AddedToken, typing.List[typing.Union[str, tokenizers.AddedToken]]]special\_tokens: bool = False ) **→** int

**Parameters**

* **new\_tokens** (str, tokenizers.AddedToken or a list of *str* or tokenizers.AddedToken) — Tokens are only added if they are not already in the vocabulary. tokenizers.AddedToken wraps a string token to let you personalize its behavior: whether this token should only match against a single word, whether this token should strip all potential whitespaces on the left side, whether this token should strip all potential whitespaces on the right side, etc.
* **special\_tokens** (bool, *optional*, defaults to False) — Can be used to specify if the token is a special token. This mostly change the normalization behavior (special tokens like CLS or [MASK] are usually not lower-cased for instance).

See details for tokenizers.AddedToken in HuggingFace tokenizers library.

**Returns**

int

Number of tokens added to the vocabulary.

Add a list of new tokens to the tokenizer class. If the new tokens are not in the vocabulary, they are added to it with indices starting from length of the current vocabulary and and will be isolated before the tokenization algorithm is applied. Added tokens and tokens from the vocabulary of the tokenization algorithm are therefore not treated in the same way.

Note, when adding new tokens to the vocabulary, you should make sure to also resize the token embedding matrix of the model so that its embedding matrix matches the tokenizer.

In order to do that, please use the [resize\_token\_embeddings()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.resize_token_embeddings) method.

Examples:

Copied

*# Let's see how to increase the vocabulary of Bert model and tokenizer*

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

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

num\_added\_toks = tokenizer.add\_tokens(["new\_tok1", "my\_new-tok2"])

print("We have added", num\_added\_toks, "tokens")

*# Notice: resize\_token\_embeddings expect to receive the full size of the new vocabulary, i.e., the length of the tokenizer.*

model.resize\_token\_embeddings(len(tokenizer))

add\_special\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L873)

( special\_tokens\_dict: typing.Dict[str, typing.Union[str, tokenizers.AddedToken]]replace\_additional\_special\_tokens = True ) **→** int

**Parameters**

* **special\_tokens\_dict** (dictionary *str* to *str* or tokenizers.AddedToken) — Keys should be in the list of predefined special attributes: [bos\_token, eos\_token, unk\_token, sep\_token, pad\_token, cls\_token, mask\_token, additional\_special\_tokens].

Tokens are only added if they are not already in the vocabulary (tested by checking if the tokenizer assign the index of the unk\_token to them).

* **replace\_additional\_special\_tokens** (bool, *optional*,, defaults to True) — If True, the existing list of additional special tokens will be replaced by the list provided in special\_tokens\_dict. Otherwise, self.\_additional\_special\_tokens is just extended. In the former case, the tokens will NOT be removed from the tokenizer’s full vocabulary - they are only being flagged as non-special tokens. Remember, this only affects which tokens are skipped during decoding, not the added\_tokens\_encoder and added\_tokens\_decoder. This means that the previous additional\_special\_tokens are still added tokens, and will not be split by the model.

**Returns**

int

Number of tokens added to the vocabulary.

Add a dictionary of special tokens (eos, pad, cls, etc.) to the encoder and link them to class attributes. If special tokens are NOT in the vocabulary, they are added to it (indexed starting from the last index of the current vocabulary).

When adding new tokens to the vocabulary, you should make sure to also resize the token embedding matrix of the model so that its embedding matrix matches the tokenizer.

In order to do that, please use the [resize\_token\_embeddings()](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel.resize_token_embeddings) method.

Using add\_special\_tokens will ensure your special tokens can be used in several ways:

* Special tokens can be skipped when decoding using skip\_special\_tokens = True.
* Special tokens are carefully handled by the tokenizer (they are never split), similar to AddedTokens.
* You can easily refer to special tokens using tokenizer class attributes like tokenizer.cls\_token. This makes it easy to develop model-agnostic training and fine-tuning scripts.

When possible, special tokens are already registered for provided pretrained models (for instance [BertTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertTokenizer) cls\_token is already registered to be :obj*’[CLS]’* and XLM’s one is also registered to be '</s>').

Examples:

Copied

*# Let's see how to add a new classification token to GPT-2*

tokenizer = GPT2Tokenizer.from\_pretrained("gpt2")

model = GPT2Model.from\_pretrained("gpt2")

special\_tokens\_dict = {"cls\_token": "<CLS>"}

num\_added\_toks = tokenizer.add\_special\_tokens(special\_tokens\_dict)

print("We have added", num\_added\_toks, "tokens")

*# Notice: resize\_token\_embeddings expect to receive the full size of the new vocabulary, i.e., the length of the tokenizer.*

model.resize\_token\_embeddings(len(tokenizer))

assert tokenizer.cls\_token == "<CLS>"

apply\_chat\_template

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L1677)

( conversation: typing.Union[typing.List[typing.Dict[str, str]], ForwardRef('Conversation')]chat\_template: typing.Optional[str] = Noneadd\_generation\_prompt: bool = Falsetokenize: bool = Truepadding: bool = Falsetruncation: bool = Falsemax\_length: typing.Optional[int] = Nonereturn\_tensors: typing.Union[str, transformers.utils.generic.TensorType, NoneType] = None\*\*tokenizer\_kwargs ) **→** List[int]

Expand 8 parameters

**Parameters**

* **conversation** (Union[List[Dict[str, str]], “Conversation”]) — A Conversation object or list of dicts with “role” and “content” keys, representing the chat history so far.
* **chat\_template** (str, *optional*) — A Jinja template to use for this conversion. If this is not passed, the model’s default chat template will be used instead.
* **add\_generation\_prompt** (bool, *optional*) — Whether to end the prompt with the token(s) that indicate the start of an assistant message. This is useful when you want to generate a response from the model. Note that this argument will be passed to the chat template, and so it must be supported in the template for this argument to have any effect.
* **tokenize** (bool, defaults to True) — Whether to tokenize the output. If False, the output will be a string.
* **padding** (bool, defaults to False) — Whether to pad sequences to the maximum length. Has no effect if tokenize is False.
* **truncation** (bool, defaults to False) — Whether to truncate sequences at the maximum length. Has no effect if tokenize is False.
* **max\_length** (int, *optional*) — Maximum length (in tokens) to use for padding or truncation. Has no effect if tokenize is False. If not specified, the tokenizer’s max\_length attribute will be used as a default.
* **return\_tensors** (str or [TensorType](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.TensorType), *optional*) — If set, will return tensors of a particular framework. Has no effect if tokenize is False. Acceptable values are:
  + 'tf': Return TensorFlow tf.Tensor objects.
  + 'pt': Return PyTorch torch.Tensor objects.
  + 'np': Return NumPy np.ndarray objects.
  + 'jax': Return JAX jnp.ndarray objects. \*\*tokenizer\_kwargs — Additional kwargs to pass to the tokenizer.

**Returns**

List[int]

A list of token ids representing the tokenized chat so far, including control tokens. This output is ready to pass to the model, either directly or via methods like generate().

Converts a Conversation object or a list of dictionaries with "role" and "content" keys to a list of token ids. This method is intended for use with chat models, and will read the tokenizer’s chat\_template attribute to determine the format and control tokens to use when converting. When chat\_template is None, it will fall back to the default\_chat\_template specified at the class level.

batch\_decode

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L3686)

( sequences: typing.Union[typing.List[int], typing.List[typing.List[int]], ForwardRef('np.ndarray'), ForwardRef('torch.Tensor'), ForwardRef('tf.Tensor')]skip\_special\_tokens: bool = Falseclean\_up\_tokenization\_spaces: bool = None\*\*kwargs ) **→** List[str]

**Parameters**

* **sequences** (Union[List[int], List[List[int]], np.ndarray, torch.Tensor, tf.Tensor]) — List of tokenized input ids. Can be obtained using the \_\_call\_\_ method.
* **skip\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not to remove special tokens in the decoding.
* **clean\_up\_tokenization\_spaces** (bool, *optional*) — Whether or not to clean up the tokenization spaces. If None, will default to self.clean\_up\_tokenization\_spaces.
* **kwargs** (additional keyword arguments, *optional*) — Will be passed to the underlying model specific decode method.

**Returns**

List[str]

The list of decoded sentences.

Convert a list of lists of token ids into a list of strings by calling decode.

decode

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L3720)

( token\_ids: typing.Union[int, typing.List[int], ForwardRef('np.ndarray'), ForwardRef('torch.Tensor'), ForwardRef('tf.Tensor')]skip\_special\_tokens: bool = Falseclean\_up\_tokenization\_spaces: bool = None\*\*kwargs ) **→** str

**Parameters**

* **token\_ids** (Union[int, List[int], np.ndarray, torch.Tensor, tf.Tensor]) — List of tokenized input ids. Can be obtained using the \_\_call\_\_ method.
* **skip\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not to remove special tokens in the decoding.
* **clean\_up\_tokenization\_spaces** (bool, *optional*) — Whether or not to clean up the tokenization spaces. If None, will default to self.clean\_up\_tokenization\_spaces.
* **kwargs** (additional keyword arguments, *optional*) — Will be passed to the underlying model specific decode method.

**Returns**

str

The decoded sentence.

Converts a sequence of ids in a string, using the tokenizer and vocabulary with options to remove special tokens and clean up tokenization spaces.

Similar to doing self.convert\_tokens\_to\_string(self.convert\_ids\_to\_tokens(token\_ids)).

encode

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L2536)

( text: typing.Union[str, typing.List[str], typing.List[int]]text\_pair: typing.Union[str, typing.List[str], typing.List[int], NoneType] = Noneadd\_special\_tokens: bool = Truepadding: typing.Union[bool, str, transformers.utils.generic.PaddingStrategy] = Falsetruncation: typing.Union[bool, str, transformers.tokenization\_utils\_base.TruncationStrategy] = Nonemax\_length: typing.Optional[int] = Nonestride: int = 0return\_tensors: typing.Union[str, transformers.utils.generic.TensorType, NoneType] = None\*\*kwargs ) **→** List[int], torch.Tensor, tf.Tensor or np.ndarray

Expand 10 parameters

**Parameters**

* **text** (str, List[str] or List[int]) — The first sequence to be encoded. This can be a string, a list of strings (tokenized string using the tokenize method) or a list of integers (tokenized string ids using the convert\_tokens\_to\_ids method).
* **text\_pair** (str, List[str] or List[int], *optional*) — Optional second sequence to be encoded. This can be a string, a list of strings (tokenized string using the tokenize method) or a list of integers (tokenized string ids using the convert\_tokens\_to\_ids method).
* **add\_special\_tokens** (bool, *optional*, defaults to True) — Whether or not to add special tokens when encoding the sequences. This will use the underlying PretrainedTokenizerBase.build\_inputs\_with\_special\_tokens function, which defines which tokens are automatically added to the input ids. This is usefull if you want to add bos or eos tokens automatically.
* **padding** (bool, str or [PaddingStrategy](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.utils.PaddingStrategy), *optional*, defaults to False) — Activates and controls padding. Accepts the following values:
  + True or 'longest': Pad to the longest sequence in the batch (or no padding if only a single sequence if provided).
  + 'max\_length': Pad to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided.
  + False or 'do\_not\_pad' (default): No padding (i.e., can output a batch with sequences of different lengths).
* **truncation** (bool, str or [TruncationStrategy](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.tokenization_utils_base.TruncationStrategy), *optional*, defaults to False) — Activates and controls truncation. Accepts the following values:
  + True or 'longest\_first': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will truncate token by token, removing a token from the longest sequence in the pair if a pair of sequences (or a batch of pairs) is provided.
  + 'only\_first': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will only truncate the first sequence of a pair if a pair of sequences (or a batch of pairs) is provided.
  + 'only\_second': Truncate to a maximum length specified with the argument max\_length or to the maximum acceptable input length for the model if that argument is not provided. This will only truncate the second sequence of a pair if a pair of sequences (or a batch of pairs) is provided.
  + False or 'do\_not\_truncate' (default): No truncation (i.e., can output batch with sequence lengths greater than the model maximum admissible input size).
* **max\_length** (int, *optional*) — Controls the maximum length to use by one of the truncation/padding parameters.

If left unset or set to None, this will use the predefined model maximum length if a maximum length is required by one of the truncation/padding parameters. If the model has no specific maximum input length (like XLNet) truncation/padding to a maximum length will be deactivated.

* **stride** (int, *optional*, defaults to 0) — If set to a number along with max\_length, the overflowing tokens returned when return\_overflowing\_tokens=True will contain some tokens from the end of the truncated sequence returned to provide some overlap between truncated and overflowing sequences. The value of this argument defines the number of overlapping tokens.
* **is\_split\_into\_words** (bool, *optional*, defaults to False) — Whether or not the input is already pre-tokenized (e.g., split into words). If set to True, the tokenizer assumes the input is already split into words (for instance, by splitting it on whitespace) which it will tokenize. This is useful for NER or token classification.
* **pad\_to\_multiple\_of** (int, *optional*) — If set will pad the sequence to a multiple of the provided value. Requires padding to be activated. This is especially useful to enable the use of Tensor Cores on NVIDIA hardware with compute capability >= 7.5 (Volta).
* **return\_tensors** (str or [TensorType](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.TensorType), *optional*) — If set, will return tensors instead of list of python integers. Acceptable values are:
  + 'tf': Return TensorFlow tf.constant objects.
  + 'pt': Return PyTorch torch.Tensor objects.
  + 'np': Return Numpy np.ndarray objects.

\*\*kwargs — Passed along to the .tokenize() method.

**Returns**

List[int]**,**torch.Tensor**,**tf.Tensor**or**np.ndarray

The tokenized ids of the text.

Converts a string to a sequence of ids (integer), using the tokenizer and vocabulary.

Same as doing self.convert\_tokens\_to\_ids(self.tokenize(text)).

push\_to\_hub

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/utils/hub.py#L753)

( repo\_id: struse\_temp\_dir: typing.Optional[bool] = Nonecommit\_message: typing.Optional[str] = Noneprivate: typing.Optional[bool] = Nonetoken: typing.Union[bool, str, NoneType] = Nonemax\_shard\_size: typing.Union[int, str, NoneType] = '5GB'create\_pr: bool = Falsesafe\_serialization: bool = Truerevision: str = Nonecommit\_description: str = None\*\*deprecated\_kwargs )

Expand 10 parameters

**Parameters**

* **repo\_id** (str) — The name of the repository you want to push your tokenizer to. It should contain your organization name when pushing to a given organization.
* **use\_temp\_dir** (bool, *optional*) — Whether or not to use a temporary directory to store the files saved before they are pushed to the Hub. Will default to True if there is no directory named like repo\_id, False otherwise.
* **commit\_message** (str, *optional*) — Message to commit while pushing. Will default to "Upload tokenizer".
* **private** (bool, *optional*) — Whether or not the repository created should be private.
* **token** (bool or str, *optional*) — The token to use as HTTP bearer authorization for remote files. If True, will use the token generated when running huggingface-cli login (stored in ~/.huggingface). Will default to True if repo\_url is not specified.
* **max\_shard\_size** (int or str, *optional*, defaults to "5GB") — Only applicable for models. The maximum size for a checkpoint before being sharded. Checkpoints shard will then be each of size lower than this size. If expressed as a string, needs to be digits followed by a unit (like "5MB"). We default it to "5GB" so that users can easily load models on free-tier Google Colab instances without any CPU OOM issues.
* **create\_pr** (bool, *optional*, defaults to False) — Whether or not to create a PR with the uploaded files or directly commit.
* **safe\_serialization** (bool, *optional*, defaults to True) — Whether or not to convert the model weights in safetensors format for safer serialization.
* **revision** (str, *optional*) — Branch to push the uploaded files to.
* **commit\_description** (str, *optional*) — The description of the commit that will be created

Upload the tokenizer files to the 🤗 Model Hub.

Examples:

Copied

from transformers import AutoTokenizer

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

*# Push the tokenizer to your namespace with the name "my-finetuned-bert".*

tokenizer.push\_to\_hub("my-finetuned-bert")

*# Push the tokenizer to an organization with the name "my-finetuned-bert".*

tokenizer.push\_to\_hub("huggingface/my-finetuned-bert")

convert\_ids\_to\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_fast.py#L369)

( ids: typing.Union[int, typing.List[int]]skip\_special\_tokens: bool = False ) **→** str or List[str]

**Parameters**

* **ids** (int or List[int]) — The token id (or token ids) to convert to tokens.
* **skip\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not to remove special tokens in the decoding.

**Returns**

str**or**List[str]

The decoded token(s).

Converts a single index or a sequence of indices in a token or a sequence of tokens, using the vocabulary and added tokens.

convert\_tokens\_to\_ids

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_fast.py#L314)

( tokens: typing.Union[str, typing.List[str]] ) **→** int or List[int]

**Parameters**

* **tokens** (str or List[str]) — One or several token(s) to convert to token id(s).

**Returns**

int**or**List[int]

The token id or list of token ids.

Converts a token string (or a sequence of tokens) in a single integer id (or a sequence of ids), using the vocabulary.

get\_added\_vocab

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_fast.py#L238)

( ) **→** Dict[str, int]

**Returns**

Dict[str, int]

The added tokens.

Returns the added tokens in the vocabulary as a dictionary of token to index.

num\_special\_tokens\_to\_add

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_fast.py#L348)

( pair: bool = False ) **→** int

**Parameters**

* **pair** (bool, *optional*, defaults to False) — Whether the number of added tokens should be computed in the case of a sequence pair or a single sequence.

**Returns**

int

Number of special tokens added to sequences.

Returns the number of added tokens when encoding a sequence with special tokens.

This encodes a dummy input and checks the number of added tokens, and is therefore not efficient. Do not put this inside your training loop.

set\_truncation\_and\_padding

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_fast.py#L398)

( padding\_strategy: PaddingStrategytruncation\_strategy: TruncationStrategymax\_length: intstride: intpad\_to\_multiple\_of: typing.Optional[int] )

**Parameters**

* **padding\_strategy** ([PaddingStrategy](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.utils.PaddingStrategy)) — The kind of padding that will be applied to the input
* **truncation\_strategy** ([TruncationStrategy](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.tokenization_utils_base.TruncationStrategy)) — The kind of truncation that will be applied to the input
* **max\_length** (int) — The maximum size of a sequence.
* **stride** (int) — The stride to use when handling overflow.
* **pad\_to\_multiple\_of** (int, *optional*) — If set will pad the sequence to a multiple of the provided value. This is especially useful to enable the use of Tensor Cores on NVIDIA hardware with compute capability >= 7.5 (Volta).

Define the truncation and the padding strategies for fast tokenizers (provided by HuggingFace tokenizers library) and restore the tokenizer settings afterwards.

The provided tokenizer has no padding / truncation strategy before the managed section. If your tokenizer set a padding / truncation strategy before, then it will be reset to no padding / truncation when exiting the managed section.

train\_new\_from\_iterator

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_fast.py#L687)

( text\_iteratorvocab\_sizelength = Nonenew\_special\_tokens = Nonespecial\_tokens\_map = None\*\*kwargs ) **→** [PreTrainedTokenizerFast](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast)

**Parameters**

* **text\_iterator** (generator of List[str]) — The training corpus. Should be a generator of batches of texts, for instance a list of lists of texts if you have everything in memory.
* **vocab\_size** (int) — The size of the vocabulary you want for your tokenizer.
* **length** (int, *optional*) — The total number of sequences in the iterator. This is used to provide meaningful progress tracking
* **new\_special\_tokens** (list of str or AddedToken, *optional*) — A list of new special tokens to add to the tokenizer you are training.
* **special\_tokens\_map** (Dict[str, str], *optional*) — If you want to rename some of the special tokens this tokenizer uses, pass along a mapping old special token name to new special token name in this argument.
* **kwargs** (Dict[str, Any], *optional*) — Additional keyword arguments passed along to the trainer from the 🤗 Tokenizers library.

**Returns**

[PreTrainedTokenizerFast](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast)

A new tokenizer of the same type as the original one, trained on text\_iterator.

Trains a tokenizer on a new corpus with the same defaults (in terms of special tokens or tokenization pipeline) as the current one.

**BatchEncoding**

classtransformers.**BatchEncoding**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L176)

( data: typing.Union[typing.Dict[str, typing.Any], NoneType] = Noneencoding: typing.Union[tokenizers.Encoding, typing.Sequence[tokenizers.Encoding], NoneType] = Nonetensor\_type: typing.Union[NoneType, str, transformers.utils.generic.TensorType] = Noneprepend\_batch\_axis: bool = Falsen\_sequences: typing.Optional[int] = None )

**Parameters**

* **data** (dict, *optional*) — Dictionary of lists/arrays/tensors returned by the \_\_call\_\_/encode\_plus/batch\_encode\_plus methods (‘input\_ids’, ‘attention\_mask’, etc.).
* **encoding** (tokenizers.Encoding or Sequence[tokenizers.Encoding], *optional*) — If the tokenizer is a fast tokenizer which outputs additional information like mapping from word/character space to token space the tokenizers.Encoding instance or list of instance (for batches) hold this information.
* **tensor\_type** (Union[None, str, TensorType], *optional*) — You can give a tensor\_type here to convert the lists of integers in PyTorch/TensorFlow/Numpy Tensors at initialization.
* **prepend\_batch\_axis** (bool, *optional*, defaults to False) — Whether or not to add a batch axis when converting to tensors (see tensor\_type above).
* **n\_sequences** (Optional[int], *optional*) — You can give a tensor\_type here to convert the lists of integers in PyTorch/TensorFlow/Numpy Tensors at initialization.

Holds the output of the [**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__), [encode\_plus()](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.PreTrainedTokenizerBase.encode_plus) and [batch\_encode\_plus()](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.PreTrainedTokenizerBase.batch_encode_plus) methods (tokens, attention\_masks, etc).

This class is derived from a python dictionary and can be used as a dictionary. In addition, this class exposes utility methods to map from word/character space to token space.

char\_to\_token

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L555)

( batch\_or\_char\_index: intchar\_index: typing.Optional[int] = Nonesequence\_index: int = 0 ) **→** int

**Parameters**

* **batch\_or\_char\_index** (int) — Index of the sequence in the batch. If the batch only comprise one sequence, this can be the index of the word in the sequence
* **char\_index** (int, *optional*) — If a batch index is provided in *batch\_or\_token\_index*, this can be the index of the word in the sequence.
* **sequence\_index** (int, *optional*, defaults to 0) — If pair of sequences are encoded in the batch this can be used to specify which sequence in the pair (0 or 1) the provided character index belongs to.

**Returns**

int

Index of the token.

Get the index of the token in the encoded output comprising a character in the original string for a sequence of the batch.

Can be called as:

* self.char\_to\_token(char\_index) if batch size is 1
* self.char\_to\_token(batch\_index, char\_index) if batch size is greater or equal to 1

This method is particularly suited when the input sequences are provided as pre-tokenized sequences (i.e. words are defined by the user). In this case it allows to easily associate encoded tokens with provided tokenized words.

char\_to\_word

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L641)

( batch\_or\_char\_index: intchar\_index: typing.Optional[int] = Nonesequence\_index: int = 0 ) **→** int or List[int]

**Parameters**

* **batch\_or\_char\_index** (int) — Index of the sequence in the batch. If the batch only comprise one sequence, this can be the index of the character in the original string.
* **char\_index** (int, *optional*) — If a batch index is provided in *batch\_or\_token\_index*, this can be the index of the character in the original string.
* **sequence\_index** (int, *optional*, defaults to 0) — If pair of sequences are encoded in the batch this can be used to specify which sequence in the pair (0 or 1) the provided character index belongs to.

**Returns**

int**or**List[int]

Index or indices of the associated encoded token(s).

Get the word in the original string corresponding to a character in the original string of a sequence of the batch.

Can be called as:

* self.char\_to\_word(char\_index) if batch size is 1
* self.char\_to\_word(batch\_index, char\_index) if batch size is greater than 1

This method is particularly suited when the input sequences are provided as pre-tokenized sequences (i.e. words are defined by the user). In this case it allows to easily associate encoded tokens with provided tokenized words.

convert\_to\_tensors

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L680)

( tensor\_type: typing.Union[str, transformers.utils.generic.TensorType, NoneType] = Noneprepend\_batch\_axis: bool = False )

**Parameters**

* **tensor\_type** (str or [TensorType](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.TensorType), *optional*) — The type of tensors to use. If str, should be one of the values of the enum [TensorType](https://huggingface.co/docs/transformers/main/en/internal/file_utils#transformers.TensorType). If None, no modification is done.
* **prepend\_batch\_axis** (int, *optional*, defaults to False) — Whether or not to add the batch dimension during the conversion.

Convert the inner content to tensors.

sequence\_ids

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L319)

( batch\_index: int = 0 ) **→** List[Optional[int]]

**Parameters**

* **batch\_index** (int, *optional*, defaults to 0) — The index to access in the batch.

**Returns**

List[Optional[int]]

A list indicating the sequence id corresponding to each token. Special tokens added by the tokenizer are mapped to None and other tokens are mapped to the index of their corresponding sequence.

Return a list mapping the tokens to the id of their original sentences:

* None for special tokens added around or between sequences,
* 0 for tokens corresponding to words in the first sequence,
* 1 for tokens corresponding to words in the second sequence when a pair of sequences was jointly encoded.

to

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L773)

( device: typing.Union[str, ForwardRef('torch.device')] ) **→** [BatchEncoding](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.BatchEncoding)

**Parameters**

* **device** (str or torch.device) — The device to put the tensors on.

**Returns**

[BatchEncoding](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.BatchEncoding)

The same instance after modification.

Send all values to device by calling v.to(device) (PyTorch only).

token\_to\_chars

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L516)

( batch\_or\_token\_index: inttoken\_index: typing.Optional[int] = None ) **→** [CharSpan](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.CharSpan)

**Parameters**

* **batch\_or\_token\_index** (int) — Index of the sequence in the batch. If the batch only comprise one sequence, this can be the index of the token in the sequence.
* **token\_index** (int, *optional*) — If a batch index is provided in *batch\_or\_token\_index*, this can be the index of the token or tokens in the sequence.

**Returns**

[CharSpan](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.CharSpan)

Span of characters in the original string, or None, if the token (e.g. ~~,~~) doesn’t correspond to any chars in the origin string.

Get the character span corresponding to an encoded token in a sequence of the batch.

Character spans are returned as a [CharSpan](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.CharSpan) with:

* **start** — Index of the first character in the original string associated to the token.
* **end** — Index of the character following the last character in the original string associated to the token.

Can be called as:

* self.token\_to\_chars(token\_index) if batch size is 1
* self.token\_to\_chars(batch\_index, token\_index) if batch size is greater or equal to 1

token\_to\_sequence

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L386)

( batch\_or\_token\_index: inttoken\_index: typing.Optional[int] = None ) **→** int

**Parameters**

* **batch\_or\_token\_index** (int) — Index of the sequence in the batch. If the batch only comprises one sequence, this can be the index of the token in the sequence.
* **token\_index** (int, *optional*) — If a batch index is provided in *batch\_or\_token\_index*, this can be the index of the token in the sequence.

**Returns**

int

Index of the word in the input sequence.

Get the index of the sequence represented by the given token. In the general use case, this method returns 0 for a single sequence or the first sequence of a pair, and 1 for the second sequence of a pair

Can be called as:

* self.token\_to\_sequence(token\_index) if batch size is 1
* self.token\_to\_sequence(batch\_index, token\_index) if batch size is greater than 1

This method is particularly suited when the input sequences are provided as pre-tokenized sequences (i.e., words are defined by the user). In this case it allows to easily associate encoded tokens with provided tokenized words.

token\_to\_word

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L425)

( batch\_or\_token\_index: inttoken\_index: typing.Optional[int] = None ) **→** int

**Parameters**

* **batch\_or\_token\_index** (int) — Index of the sequence in the batch. If the batch only comprise one sequence, this can be the index of the token in the sequence.
* **token\_index** (int, *optional*) — If a batch index is provided in *batch\_or\_token\_index*, this can be the index of the token in the sequence.

**Returns**

int

Index of the word in the input sequence.

Get the index of the word corresponding (i.e. comprising) to an encoded token in a sequence of the batch.

Can be called as:

* self.token\_to\_word(token\_index) if batch size is 1
* self.token\_to\_word(batch\_index, token\_index) if batch size is greater than 1

This method is particularly suited when the input sequences are provided as pre-tokenized sequences (i.e., words are defined by the user). In this case it allows to easily associate encoded tokens with provided tokenized words.

tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L301)

( batch\_index: int = 0 ) **→** List[str]

**Parameters**

* **batch\_index** (int, *optional*, defaults to 0) — The index to access in the batch.

**Returns**

List[str]

The list of tokens at that index.

Return the list of tokens (sub-parts of the input strings after word/subword splitting and before conversion to integer indices) at a given batch index (only works for the output of a fast tokenizer).

word\_ids

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L367)

( batch\_index: int = 0 ) **→** List[Optional[int]]

**Parameters**

* **batch\_index** (int, *optional*, defaults to 0) — The index to access in the batch.

**Returns**

List[Optional[int]]

A list indicating the word corresponding to each token. Special tokens added by the tokenizer are mapped to None and other tokens are mapped to the index of their corresponding word (several tokens will be mapped to the same word index if they are parts of that word).

Return a list mapping the tokens to their actual word in the initial sentence for a fast tokenizer.

word\_to\_chars

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L596)

( batch\_or\_word\_index: intword\_index: typing.Optional[int] = Nonesequence\_index: int = 0 ) **→** CharSpan or List[CharSpan]

**Parameters**

* **batch\_or\_word\_index** (int) — Index of the sequence in the batch. If the batch only comprise one sequence, this can be the index of the word in the sequence
* **word\_index** (int, *optional*) — If a batch index is provided in *batch\_or\_token\_index*, this can be the index of the word in the sequence.
* **sequence\_index** (int, *optional*, defaults to 0) — If pair of sequences are encoded in the batch this can be used to specify which sequence in the pair (0 or 1) the provided word index belongs to.

**Returns**

CharSpan**or**List[CharSpan]

Span(s) of the associated character or characters in the string. CharSpan are NamedTuple with:

* start: index of the first character associated to the token in the original string
* end: index of the character following the last character associated to the token in the original string

Get the character span in the original string corresponding to given word in a sequence of the batch.

Character spans are returned as a CharSpan NamedTuple with:

* start: index of the first character in the original string
* end: index of the character following the last character in the original string

Can be called as:

* self.word\_to\_chars(word\_index) if batch size is 1
* self.word\_to\_chars(batch\_index, word\_index) if batch size is greater or equal to 1

word\_to\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L463)

( batch\_or\_word\_index: intword\_index: typing.Optional[int] = Nonesequence\_index: int = 0 ) **→** ([TokenSpan](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.TokenSpan), *optional*)

**Parameters**

* **batch\_or\_word\_index** (int) — Index of the sequence in the batch. If the batch only comprises one sequence, this can be the index of the word in the sequence.
* **word\_index** (int, *optional*) — If a batch index is provided in *batch\_or\_token\_index*, this can be the index of the word in the sequence.
* **sequence\_index** (int, *optional*, defaults to 0) — If pair of sequences are encoded in the batch this can be used to specify which sequence in the pair (0 or 1) the provided word index belongs to.

**Returns**

**(**[TokenSpan](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.TokenSpan)**, *optional*)**

Span of tokens in the encoded sequence. Returns None if no tokens correspond to the word. This can happen especially when the token is a special token that has been used to format the tokenization. For example when we add a class token at the very beginning of the tokenization.

Get the encoded token span corresponding to a word in a sequence of the batch.

Token spans are returned as a [TokenSpan](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.TokenSpan) with:

* **start** — Index of the first token.
* **end** — Index of the token following the last token.

Can be called as:

* self.word\_to\_tokens(word\_index, sequence\_index: int = 0) if batch size is 1
* self.word\_to\_tokens(batch\_index, word\_index, sequence\_index: int = 0) if batch size is greater or equal to 1

This method is particularly suited when the input sequences are provided as pre-tokenized sequences (i.e. words are defined by the user). In this case it allows to easily associate encoded tokens with provided tokenized words.

words

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L343)

( batch\_index: int = 0 ) **→** List[Optional[int]]

**Parameters**

* **batch\_index** (int, *optional*, defaults to 0) — The index to access in the batch.

**Returns**

List[Optional[int]]

A list indicating the word corresponding to each token. Special tokens added by the tokenizer are mapped to None and other tokens are mapped to the index of their corresponding word (several tokens will be mapped to the same word index if they are parts of that word).

Return a list mapping the tokens to their actual word in the initial sentence for a fast tokenizer.