[![Hugging Face's logo](/front/assets/huggingface_logo-noborder.svg) Hugging

Face](/)

  * [Models](/models)

  * [Datasets](/datasets)

  * [Spaces](/spaces)

  * [Posts](/posts)

  * [Docs](/docs)

  * [Enterprise](/enterprise)

  * [Pricing](/pricing)

  * * * * *

  * [Log In](/login)

  * [Sign Up](/join)

#

[](/Alib

aba-

NLP)

[Alibaba-NLP](/Alibaba-NLP)

/

[gte-Qwen2-1.5B-instruct](/Alibaba-NLP/gte-Qwen2-1.5B-instruct)

like 158

Follow

Alibaba-NLP 279

[Sentence Similarity](/models?pipeline_tag=sentence-similarity)[sentence-transformers](/models?library=sentence-transformers)[Safetensors](/models?library=safetensors)[Transformers](/models?library=transformers)[qwen2](/models?other=qwen2)[text-generation](/models?other=text-generation)[mteb](/models?other=mteb)[Qwen2](/models?other=Qwen2)[custom_code](/models?other=custom_code)[Eval Results](/models?other=model-index)[text-generation-inference](/models?other=text-generation-inference)[text-embeddings-inference](/models?other=text-embeddings-inference)[Inference Endpoints](/models?other=endpoints_compatible)

arxiv: 2308.03281

License: apache-2.0

[Model card](/Alibaba-NLP/gte-Qwen2-1.5B-instruct)[Files Files and versions](/Alibaba-NLP/gte-Qwen2-1.5B-instruct/tree/main)[Community 30](/Alibaba-NLP/gte-Qwen2-1.5B-instruct/discussions)

Train

Deploy

Use this model

main

[gte-Qwen2-1.5B-instruct](/Alibaba-NLP/gte-Qwen2-1.5B-instruct/tree/main) /

tokenization_qwen.py

zhangyanzhao.zyz

init model

[a3fa269](/Alibaba-NLP/gte-Qwen2-1.5B-instruct/commit/a3fa2698dec82d2860480fa4286382847b58aa12) 7 months ago

[raw](/Alibaba-NLP/gte-Qwen2-1.5B-instruct/raw/main/tokenization_qwen.py)

Copy download link

[history](/Alibaba-NLP/gte-Qwen2-1.5B-instruct/commits/main/tokenization_qwen.py)[blame](/Alibaba-NLP/gte-Qwen2-1.5B-instruct/blame/main/tokenization_qwen.py)[contribute](/Alibaba-NLP/gte-Qwen2-1.5B-instruct/edit/main/tokenization_qwen.py)[delete](/Alibaba-NLP/gte-Qwen2-1.5B-instruct/delete/main/tokenization_qwen.py)

Safe

10.8 kB

```
|
---|---
| from typing import List, Optional
| from transformers.models.qwen2.tokenization_qwen2 import Qwen2Tokenizer as
OriginalQwen2Tokenizer
| from transformers.models.qwen2.tokenization_qwen2_fast import
Qwen2TokenizerFast as OriginalQwen2TokenizerFast
| from tokenizers import processors
|
| VOCAB_FILES_NAMES = {
|  "vocab_file": "vocab.json",
|  "merges_file": "merges.txt",
|  "tokenizer_file": "tokenizer.json",
| }
|
| class Qwen2Tokenizer(OriginalQwen2Tokenizer):
|  """
|  Construct a Qwen2 tokenizer. Based on byte-level Byte-Pair-Encoding.
|
|  Same with GPT2Tokenizer, this tokenizer has been trained to treat spaces
like parts of the tokens so a word will
|  be encoded differently whether it is at the beginning of the sentence
(without space) or not:
```

|

| ```python

| >>> from transformers import Qwen2Tokenizer

|

| >>> tokenizer = Qwen2Tokenizer.from_pretrained("Qwen/Qwen-tokenizer")

| >>> tokenizer("Hello world")["input_ids"]

| [9707, 1879]

|

| >>> tokenizer(" Hello world")["input_ids"]

| [21927, 1879]

| ```

| This is expected.

|

| You should not use GPT2Tokenizer instead, because of the different

pretokenization rules.

|

| This tokenizer inherits from [`PreTrainedTokenizer`] which contains most of

the main methods. Users should refer to

| this superclass for more information regarding those methods.

|

| Args:

| vocab_file (`str`):

| Path to the vocabulary file.

| merges_file (`str`):

| Path to the merges file.

| errors (`str`, *optional*, defaults to `"replace"`):

| Paradigm to follow when decoding bytes to UTF-8. See

|

[bytes.decode](https://docs.python.org/3/library/stdtypes.html#bytes.decode)

for more information.

| unk_token (`str`, *optional*, defaults to `"<|endoftext|>"`):

| The unknown token. A token that is not in the vocabulary cannot be

converted to an ID and is set to be this

| token instead.

| bos_token (`str`, *optional*):

| The beginning of sequence token. Not applicable for this tokenizer.

| eos_token (`str`, *optional*, defaults to `"<|endoftext|>"`):

| The end of sequence token.

| pad_token (`str`, *optional*, defaults to `"<|endoftext|>"`):

| The token used for padding, for example when batching sequences of

different lengths.

| clean_up_tokenization_spaces (`bool`, *optional*, defaults to `False`):

| Whether or not the model should cleanup the spaces that were added when

splitting the input text during the

| tokenization process. Not applicable to this tokenizer, since tokenization

does not add spaces.

| 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 `<|endoftext|>` is the

`eos_token`, then `tokenizer.tokenize("<|endoftext|>") =

| ['<|endoftext|>`]. Otherwise, if `split_special_tokens=True`, then

`tokenizer.tokenize("<|endoftext|>")` will be give `['<',

| '|', 'endo', 'ft', 'ext', '|', '>']`. This argument is only supported for

`slow` tokenizers for the moment.

| add_eos_token (`bool`, *optional*, defaults to `False`):

| Whether or not to add an `eos_token` at the end of sequences.

| """

|

| def __init__(

| self,

| vocab_file,

| merges_file,

| errors="replace",

| unk_token="<|endoftext|>",

| bos_token=None,

| eos_token="<|endoftext|>",

| pad_token="<|endoftext|>",

| clean_up_tokenization_spaces=False,

| split_special_tokens=False,

| add_eos_token=False,

| **kwargs,

|):

| # The add_eos_token code was inspired by the LlamaTokenizer

| self.add_eos_token = add_eos_token

|

| super().__init__(

| vocab_file=vocab_file,

| merges_file=merges_file,

| errors=errors,

| unk_token=unk_token,

```
|  bos_token=bos_token,
|  eos_token=eos_token,
|  pad_token=pad_token,
|  clean_up_tokenization_spaces=clean_up_tokenization_spaces,
|  split_special_tokens=split_special_tokens,
|  add_eos_token=add_eos_token,
|  **kwargs,
|  )
|
|  def build_inputs_with_special_tokens(self, token_ids_0, token_ids_1=None):
|  eos_token_id = [self.eos_token_id] if self.add_eos_token else []
|
|  output = token_ids_0 + eos_token_id
|
|  if token_ids_1 is not None:
|  output = output + token_ids_1 + eos_token_id
|
|  return output
|
|  def get_special_tokens_mask(
|  self, token_ids_0: List[int], token_ids_1: Optional[List[int]] = None,
already_has_special_tokens: bool = False
|  ) -> List[int]:
|  """
|  Retrieve sequence ids from a token list that has no special tokens added.
This method is called when adding
|  special tokens using the tokenizer `prepare_for_model` method.
```

|

| Args:

| token_ids_0 (`List[int]`):

| List of IDs.

| token_ids_1 (`List[int]`, *optional*):

| Optional second list of IDs for sequence pairs.

| already_has_special_tokens (`bool`, *optional*, defaults to `False`):

| Whether or not the token list is already formatted with special tokens for

the model.

|

| Returns:

| `List[int]`: A list of integers in the range [0, 1]: 1 for a special token,

0 for a sequence token.

| """

| if already_has_special_tokens:

| return super().get_special_tokens_mask(

| token_ids_0=token_ids_0, token_ids_1=token_ids_1,

already_has_special_tokens=True

|)

|

| eos_token_id = [1] if self.add_eos_token else []

|

| if token_ids_1 is None:

| return ([0] * len(token_ids_0)) + eos_token_id

| return (

| ([0] * len(token_ids_0))

| \+ eos_token_id

| \+ ([0] * len(token_ids_1))

| \+ eos_token_id

|)

|

| def create_token_type_ids_from_sequences(

| self, token_ids_0: List[int], token_ids_1: Optional[List[int]] = None

|) -> List[int]:

| """

| Creates a mask from the two sequences passed to be used in a sequence-pair

classification task. An ALBERT

| sequence pair mask has the following format:

|

| ```

| 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1

| | first sequence | second sequence |

| ```

|

| if token_ids_1 is None, only returns the first portion of the mask (0s).

|

| Args:

| token_ids_0 (`List[int]`):

| List of ids.

| token_ids_1 (`List[int]`, *optional*):

| Optional second list of IDs for sequence pairs.

|

| Returns:

| `List[int]`: List of [token type IDs](../glossary#token-type-ids) according

to the given sequence(s).

| """

| eos_token_id = [self.eos_token_id] if self.add_eos_token else []

|

| output = [0] * len(token_ids_0 + eos_token_id)

|

| if token_ids_1 is not None:

| output += [1] * len(token_ids_1 + eos_token_id)

|

| return output

|

| class Qwen2TokenizerFast(OriginalQwen2TokenizerFast):

| """

| Construct a "fast" Qwen2 tokenizer (backed by HuggingFace's *tokenizers*

library). Based on byte-level

| Byte-Pair-Encoding.

|

| Same with GPT2Tokenizer, this tokenizer has been trained to treat spaces

like parts of the tokens so a word will

| be encoded differently whether it is at the beginning of the sentence

(without space) or not:

|

| ```python

| >>> from transformers import Qwen2TokenizerFast

|

| >>> tokenizer = Qwen2TokenizerFast.from_pretrained("Qwen/Qwen-tokenizer")

| >>> tokenizer("Hello world")["input_ids"]

| [9707, 1879]

|

| >>> tokenizer(" Hello world")["input_ids"]

| [21927, 1879]

| ```

| This is expected.

|

| This tokenizer inherits from [`PreTrainedTokenizerFast`] which contains

most of the main methods. Users should

| refer to this superclass for more information regarding those methods.

|

| Args:

| vocab_file (`str`, *optional*):

| Path to the vocabulary file.

| merges_file (`str`, *optional*):

| Path to the merges file.

| tokenizer_file (`str`, *optional*):

| Path to [tokenizers](https://github.com/huggingface/tokenizers) file

(generally has a .json extension) that

| contains everything needed to load the tokenizer.

| unk_token (`str`, *optional*, defaults to `"<|endoftext|>"`):

| The unknown token. A token that is not in the vocabulary cannot be

converted to an ID and is set to be this

| token instead. Not applicable to this tokenizer.

| bos_token (`str`, *optional*):

| The beginning of sequence token. Not applicable for this tokenizer.

| eos_token (`str`, *optional*, defaults to `"<|endoftext|>"`):

| The end of sequence token.

| pad_token (`str`, *optional*, defaults to `"<|endoftext|>"`):

| The token used for padding, for example when batching sequences of

different lengths.

| add_eos_token (`bool`, *optional*, defaults to `False`):

| Whether or not to add an `eos_token` at the end of sequences.

| """

|

| slow_tokenizer_class = Qwen2Tokenizer

| padding_side = "left"

|

| def __init__(

| self,

| vocab_file=None,

| merges_file=None,

| tokenizer_file=None,

| unk_token="<|endoftext|>",

| bos_token=None,

| eos_token="<|endoftext|>",

| pad_token="<|endoftext|>",

| add_eos_token=False,

| **kwargs,

|):

| super().__init__(

| vocab_file=vocab_file,

| merges_file=merges_file,

| tokenizer_file=tokenizer_file,

```
|  unk_token=unk_token,
|  bos_token=bos_token,
|  eos_token=eos_token,
|  pad_token=pad_token,
|  **kwargs,
|  )
|
|  self._add_eos_token = add_eos_token
|  self.update_post_processor()
|
|  def update_post_processor(self):
|  """
|  Updates the underlying post processor with the current `eos_token`.
|  """
|  eos = self.eos_token
|  eos_token_id = self.eos_token_id
|  if eos is None and self.add_eos_token:
|  raise ValueError("add_eos_token = True but eos_token = None")
|
|  single = f"$A:0{(' '+eos+':0') if self.add_eos_token else ''}"
|  pair = f"{single} $B:1{(' '+eos+':1') if self.add_eos_token else ''}"
|
|  special_tokens = []
|  if self.add_eos_token:
|  special_tokens.append((eos, eos_token_id))
|  self._tokenizer.post_processor = processors.TemplateProcessing(
|  single=single, pair=pair, special_tokens=special_tokens
```

|)

|

| @property

| def add_eos_token(self):

| return self._add_eos_token