Tokenization

> “□□□□□□□□ “tokenization”
> □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

Qwen-7B□□ UTF-8□□□□□□ BPE tokenization□□□□□□□ `tiktoken`□□□□□□□□□□□□□□
Qwen-7B□□□□ token□□□□ BPE□ `bytes`□□□□□ token□□□□□□ `str`□□□□□ token□

```
from transformers import AutoTokenizer


tokenizer = AutoTokenizer.from_pretrained('Qwen/Qwen-7B', trust_remote_code=True)
```

??token

□□ token□□ BPE□□□ UTF-8□□□□□□□□□□□□□□□□□□□

```
[b' \xe6\xa0', b'\xb9']


>>> b' \xe6\xa0\xb9'.decode("utf-8", errors='replace')
' 根'
```

`bytes` 类型的普通 token到 id的映射可以通过 `tokenizer.get_vocab()` 获取。尚不支持也不推荐向 tokenizer 增加普通 token。

??token

特殊 token用以给模型传递特殊信号，如到达文本末尾。理论上，输入文本中不包含特殊 token，它们仅在 tokenization后由开发者手动加入。特殊 token的字面表达，如表示文本结束的 `<|endoftext|>`，仅便于指代特殊 token，不意味着它们在输入文本空间中。

目前，训练中使用的、已经有固定含义的、不应做它用的特殊 token，Qwen-7B中有 `<|endoftext|>`，Qwen-7B-Chat中有 `<|endoftext|>`、`<|im_start|>` 以及 `<|im_end|>`。但词表中也留有供扩展的特殊 token位，可用 `<|extra_0|>` 到 `<|extra_204|>` 来指代。`str` 类型的特殊 token字面表达到 id的映射，可以通过 `tokenizer.special_tokens` 获取。

对于提供的模型参数 (Qwen-7B和 Qwen-7B-Chat)而言，诸如 `bos`、`eos`、`unk`、`pad`、`mask`、`sep` 等的特殊 token的概念并不适用。特例是 `pad`，由于这个 token理论上并不参与模型计算，所以可以使用任意 token 表达这一概念。但保险起见，目前可在 tokenizer初始化时设定的特殊 token，仅可使用已知的特殊 token 字面表达，即 `<|endoftext|>`、`<|im_start|>`、`<|im_end|>` 和 `<|extra_0|>` 到 `<|extra_204|>`。对于微调或者其它需要这些 token才能运行的框架，可以如下配置

```
from transformers import AutoTokenizer


tokenizer = AutoTokenizer.from_pretrained('Qwen/Qwen-7B', trust_remote_code=True,
pad_token='<|endoftext|>')
```

> 注意：对于提供的训练好的模型，设置诸如 `bos`、`eos`、`unk` 之类的没有意义，即模型不需要这些概念。如果设置了这些 token，但没有相应的微调这些 token以让模型理解其含义，未知行为可能被触发。特别时，不应混淆 `<|endoftext|>` 和 `eos` 的概念，除非应用场景中它们的实际含义是一致的，即句子末尾等价于文本末尾。

注入攻击防御

由于特殊 token和普通 token概念上的差异，如果输入文本中含有特殊 token的字面表达该如何处理？以下面文本为例

```
print("<|endoftext|>")
```

其正确的 tokenization为

```
ids:[1350, 9639, 91, 8691, 723, 427, 91, 82598]
tokens: [b'print', b'("<', b'|', b'endo', b'ft', b'ext', b'|', b'>")']
```

不是

```
ids: [1350, 445, 151643, 899]
tokens: [b'print', b'("', '<|endoftext|>', b'")']
```

默认行为曾是正确的，即输入文本中任何字符一律按普通 token处理，特殊 token应由开发者在 tokenization 人工处理。 然后，这与社区中的实践似有差异，为开发者复用代码增加了额外适配步骤。

默认行为已被调整为从输入文本中解析特殊 token的字面表达。 如需启用注入攻击防御，请传入参数 `allowed_special=set()`:

```
>>> tokenizer('print("<|endoftext|>")', allowed_special=set())
{'input_ids': [1350, 9639, 91, 8691, 723, 427, 91, 82598], 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1]}
```

这一行为可以更精细的调控，将 `allowed_special` 设计为 `str` 的集合即可:

```
>>> tokenizer('print("<|extra_0|>")<|endoftext|>', allowed_special={'<|endoftext|>'})
{'input_ids': [1350, 9639, 91, 15460, 62, 15, 91, 82598, 151643], 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

如果希望输入中遇到特殊 token的字面表达时，获得更直接的提醒，通过配置 `disallowed_special` 可以让 tokenizer直接触发异常:

```
>>> tokenizer('print("<|extra_0|>")<|endoftext|>', allowed_special={'<|endoftext|>'},
disallowed_special=('<|extra_0|>', ))
...
ValueError: Encountered text corresponding to disallowed special token '<|extra_0|>'.
If you want this text to be encoded as a special token, pass it to `allowed_special`, e.g.
`allowed_special={'<|extra_0|>', ...}`.
If you want this text to be encoded as normal text, disable the check for this token by
passing `disallowed_special=(enc.special_tokens_set - {'<|extra_0|>'})`.
To disable this check for all special tokens, pass `disallowed_special=()`.
```

更多关于 `allowed_special` 和 `disallowed_special` 的信息 ，请参阅 `tiktoken` 代码__.

新的默认行为与以下设定等价

```
>>> tokenizer('print("<|endoftext|>")', allowed_special="all", disallowed_special=())
{'input_ids': [1350, 445, 151643, 899], 'token_type_ids': [0, 0, 0, 0], 'attention_mask': [1,
1, 1, 1]}
```

????

> 特别提醒：请仔细阅读本部分的说明，理解每一步操作，并承担可能的后果。
>
> 由于词表扩展部分由您提供，产出方式的差异可能导致特定的不兼容情况，请审慎操作。

Qwen系列模型的 tokenizer基于 BPE方案提取文本中的 token。从 UTF-8 编码的字节开始（每个字节都可以是一个 token），两两 token合并成为新 token，直至不能再合并出新的 token为止。由于词表同时还记录了 token的合并方式，直接向词表中添加词可能对 Qwen的 tokenizer并不适用，即通过已有的 token可能合并不出来您添加词。

因而，请参照以下步骤获得合并信息：

1. 准备一个纯文本文件，例如名为 `qwen_extra_vocab.txt`，每行一个待添加的词和它的频率，中间用制表符 `\t` 分隔。

   以下是一个文件的例子：

   ```
   我是一只猫	20
   你是一只猫	10
   他是一只猫	5
   一只	200
   一只猫	100
   夸张的 比喻手法	20
   ```

   频率是必需的，用来计算合并的优先级。
2. 准备基础的词表文件，例如 `qwen.tiktoken`，并确认新加入 token的起始索引。

   Qwen模型词表中有 151,643个普通 token，有 208个特殊 token。简单起见，起始索引可以设置为 151,851（默认值）。您可以覆写不起效的特殊 token，但您需要相应的修改 tokenizer代码。
3. 运行以下命令：

   ```
   python add_merges.py qwen.tiktoken qwen_extra.tiktoken qwen_extra_vocab.txt
   ```

   `add_merges.py` 代码在 GitHub存储库中。基于提供的 `qwen_extra_vocab.txt`，该脚本将学习新的 token合并方式。新 token及其索引将存储在 `qwen_extra.tiktoken` 文件中。您可以视情况修改有关路径。

   由于是纯 Python实现，如果您添加了非常多的词，预期会花费较多时间。

   请注意，由于预切分，有些词是无法作为 token加入的。如果您添加了这些词，您会收到警告：

```
WARNING - 夸张的比喻手法 would be pre-tokenized to ['夸张的', ' 比喻手法'], and thus cannot be added to vocabulary
WARNING - word 一只 is already a token b'\xe4\xb8\x80\xe5\x8f\xaa', skipping
INFO - number of existing merges: 151643
INFO - number of words for expanding: 4
DEBUG - (b'\xe4\xb8\x80\xe5\x8f\xaa', b'\xe7\x8c\xab') (一只猫) is selected as the next merge with freq 100
DEBUG - (b'\xe5\x8f\xaa', b'\xe7\x8c\xab') (只猫) is selected as the next merge with freq 35
DEBUG - (b'\xe6\x98\xaf\xe4\xb8\x80', b'\xe5\x8f\xaa\xe7\x8c\xab') (是一只猫) is selected as the next merge with freq 35
DEBUG - (b'\xe6\x88\x91', b'\xe6\x98\xaf\xe4\xb8\x80\xe5\x8f\xaa\xe7\x8c\xab') (我是一只猫) is selected as the next merge with freq 20
DEBUG - (b'\xe4\xbd\xa0', b'\xe6\x98\xaf\xe4\xb8\x80\xe5\x8f\xaa\xe7\x8c\xab') (你是一只猫) is selected as the next merge with freq 10
DEBUG - (b'\xe4\xbb\x96', b'\xe6\x98\xaf\xe4\xb8\x80\xe5\x8f\xaa\xe7\x8c\xab') (他是一只猫) is selected as the next merge with freq 5
INFO - number of newly learned merges: 6
```

`qwen_extra.tiktoken`会包含以下内容:

```
5LiA5Y+q54yr 151851
5Y+q54yr 151852
5piv5LiA5Y+q54yr 151853
5oiR5piv5LiA5Y+q54yr 151854
5L2g5piv5LiA5Y+q54yr 151855
5LuW5piv5LiA5Y+q54yr 151856
```

您可以按如下方式使用扩展后的词表:

```
from transformers import AutoTokenizer

>>> tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen-7B", trust_remote_code=True, extra_vocab_file="qwen_extra.tiktoken")

>>> len(tokenizer)
151857

>>> tokenizer("我是一只猫")
```

```
{'input_ids': [151854], 'token_type_ids': [0], 'attention_mask': [1]}
```

注意：您需要使用2023年10月8日后的tokenizer代码才能传递`extra_vocab_file`参数。如是其它情况，您可以将`qwen_extra.tiktoken`内容复制粘贴到`qwen.tiktoken`内容后面。

您需要微调模型才能使新的token发挥作用。

????

Qwen的tokenizer是直接从UTF-8编码的字节序列开始处理的，这与其它tokenizer比如SentencePiece是很不一样的。SentencePiece是从Unicode码位（可以理解为一个字符）开始处理，遇到未登录的再用UTF-8编码成字节。

从字节开始的一个潜在问题是如果频率信息不够准确，比如频率信息是在很少数据上统计得到的，Unicode码位按UTF-8编码成字节后的边界可能会出现差错。

理论上，如果模型微调数据量不足，使用扩展后的词表也可能出现意外问题。

举个例子（非实际情况），对于`一只`的UTF-8字节序列`b'\xe4\xb8\x80\xe5\x8f\xaa'`，中间两个字节`b'\x80\xe5'`可能会先合并为一个token，跨越了`一`(`b'\xe4\xb8\x80'`)和`只`(`b'\xe5\x8f\xaa'`)的码位边界。这对于已登录token不会有什么影响（最后总会合并为`一只`），但对于未登录的，可能会产生一些不同寻常的合并/token。这些token序列可能对于预训练模型是陌生的。

我们的建议是保险起见，您最好先收集待添加词中的所有Unicode码位，然后单独指定它们的频率大于其所构成词的频率之和。不过由于Qwen的tokenizer已包含了大多数中文字，对于中文词的话，不添加中文字的频率，大部分情况下是可行的。

您可能已经发现了，在提供的例子中，`一只`已经是登录过的token了，但`只猫`还是学习成为了一个新token，出现了“交叉”。原因是在Qwen中`是一`也是一个已知token，且其频率/优先级比`一只`要高，因而对于`是|一|只|猫`这个片段，合并的次序是`是一|只|猫 -> 是一|只猫 -> 是一只猫`（省略UTF-8字节级别的合并）。

这是常规BPE的特性，其完全基于分布，并不知道哪些字节可以构成合法的Unicode码位、合法的字符或是词。

副产物是一段文本在不同的上下文下可能会有不同的tokenize结果，对于仅包含ASCII字符的文本同样如此。

```
>>> tokenizer.tokenize("Panda")
[b'P', b'anda']

>>> tokenizer.tokenize(" Panda")
[b' Panda']

>>> tokenizer.tokenize("Pandas")
[b'P', b'andas']

>>> tokenizer.tokenize(" Pandas")
```

```
[b' Pand', b'as']
```

这仅说明在用于学习BPE的数据中，这样的组合是更高频的。如果您有海量的训练语料，这并不会是个问题。

Revision #2
Created 2025-01-11 09:44:04 UTC by Colin
Updated 2026-04-29 07:34:30 UTC by Colin