

Next, we will test the tokenizer further on text that contains unknown words and discuss additional special tokens that can be used to provide further context for an LLM during training.

2.4 Adding special context tokens

We need to modify the tokenizer to handle unknown words. We also need to address the usage and addition of special context tokens that can enhance a model’s understanding of context or other relevant information in the text. These special tokens can include markers for unknown words and document boundaries, for example. In particular, we will modify the vocabulary and tokenizer, `SimpleTokenizerV2`, to support two new tokens, `<|unk|>` and `<|endoftext|>`, as illustrated in figure 2.9.

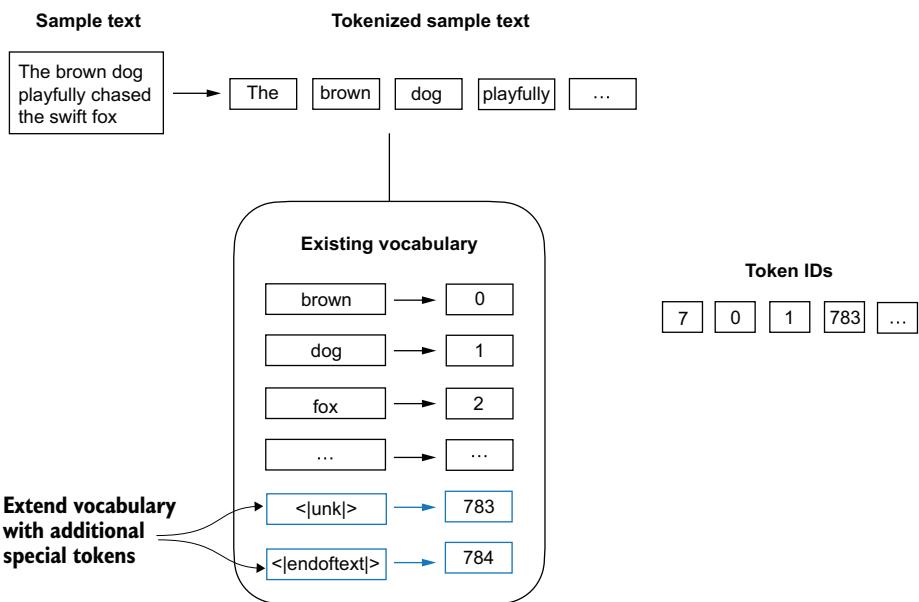


Figure 2.9 We add special tokens to a vocabulary to deal with certain contexts. For instance, we add an `<|unk|>` token to represent new and unknown words that were not part of the training data and thus not part of the existing vocabulary. Furthermore, we add an `<|endoftext|>` token that we can use to separate two unrelated text sources.

We can modify the tokenizer to use an `<|unk|>` token if it encounters a word that is not part of the vocabulary. Furthermore, we add a token between unrelated texts. For example, when training GPT-like LLMs on multiple independent documents or books, it is common to insert a token before each document or book that follows a previous text source, as illustrated in figure 2.10. This helps the LLM understand that although these text sources are concatenated for training, they are, in fact, unrelated.

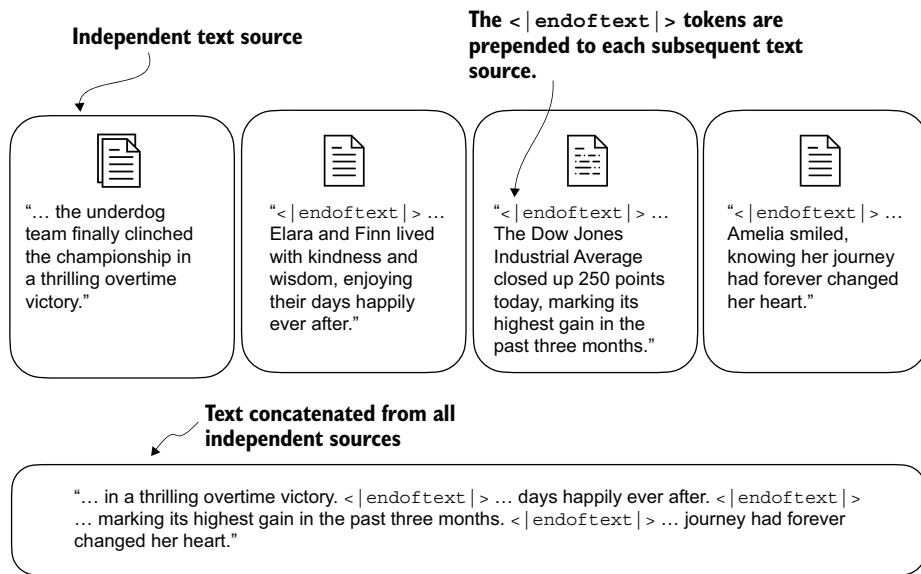


Figure 2.10 When working with multiple independent text source, we add <| endoftext |> tokens between these texts. These <| endoftext |> tokens act as markers, signaling the start or end of a particular segment, allowing for more effective processing and understanding by the LLM.

Let's now modify the vocabulary to include these two special tokens, <unk> and <| endoftext |>, by adding them to our list of all unique words:

```
all_tokens = sorted(list(set(preprocessed)))
all_tokens.extend(["<| endoftext |>", "<| unk |>"])
vocab = {token:integer for integer,token in enumerate(all_tokens)}

print(len(vocab.items()))
```

Based on the output of this print statement, the new vocabulary size is 1,132 (the previous vocabulary size was 1,130).

As an additional quick check, let's print the last five entries of the updated vocabulary:

```
for i, item in enumerate(list(vocab.items()) [-5:]):
    print(item)
```

The code prints

```
('younger', 1127)
('your', 1128)
('yourself', 1129)
('<| endoftext |>', 1130)
('<| unk |>', 1131)
```

Based on the code output, we can confirm that the two new special tokens were indeed successfully incorporated into the vocabulary. Next, we adjust the tokenizer from code listing 2.3 accordingly as shown in the following listing.

Listing 2.4 A simple text tokenizer that handles unknown words

```
class SimpleTokenizerV2:
    def __init__(self, vocab):
        self.str_to_int = vocab
        self.int_to_str = { i:s for s,i in vocab.items() }

    def encode(self, text):
        preprocessed = re.split(r'([.,;?!"]|--|\s)', text)
        preprocessed = [
            item.strip() for item in preprocessed if item.strip()
        ]
        preprocessed = [item if item in self.str_to_int
                       else "<|unk|>" for item in preprocessed]

        ids = [self.str_to_int[s] for s in preprocessed]
        return ids

    def decode(self, ids):
        text = " ".join([self.int_to_str[i] for i in ids])
        text = re.sub(r'\s+([.,;?!"])', r'\1', text)
        return text
```

Compared to the `SimpleTokenizerV1` we implemented in listing 2.3, the new `SimpleTokenizerV2` replaces unknown words with `<|unk|>` tokens.

Let's now try this new tokenizer out in practice. For this, we will use a simple text sample that we concatenate from two independent and unrelated sentences:

```
text1 = "Hello, do you like tea?"
text2 = "In the sunlit terraces of the palace."
text = "<|endoftext|> ".join((text1, text2))
print(text)
```

The output is

```
Hello, do you like tea? <|endoftext|> In the sunlit terraces of
the palace.
```

Next, let's tokenize the sample text using the `SimpleTokenizerV2` on the vocab we previously created in listing 2.2:

```
tokenizer = SimpleTokenizerV2(vocab)
print(tokenizer.encode(text))
```