

The resulting output is:

```
['Hello', ',', 'world', '.', 'Is', 'this', '--', 'a', 'test', '?']
```

As we can see based on the results summarized in figure 2.5, our tokenization scheme can now handle the various special characters in the text successfully.

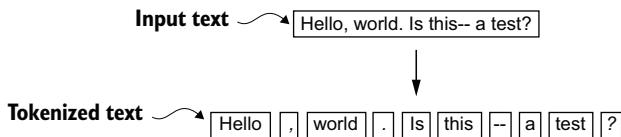


Figure 2.5 The tokenization scheme we implemented so far splits text into individual words and punctuation characters. In this specific example, the sample text gets split into 10 individual tokens.

Now that we have a basic tokenizer working, let's apply it to Edith Wharton's entire short story:

```
preprocessed = re.split(r'([.,;?!()\\"]|--|\\s)', raw_text)
preprocessed = [item.strip() for item in preprocessed if item.strip()]
print(len(preprocessed))
```

This print statement outputs 4690, which is the number of tokens in this text (without whitespaces). Let's print the first 30 tokens for a quick visual check:

```
print(preprocessed[:30])
```

The resulting output shows that our tokenizer appears to be handling the text well since all words and special characters are neatly separated:

```
['I', 'HAD', 'always', 'thought', 'Jack', 'Gisburn', 'rather', 'a',
'cheap', 'genius', '--', 'though', 'a', 'good', 'fellow', 'enough',
--, 'so', 'it', 'was', 'no', 'great', 'surprise', 'to', 'me', 'to',
'hear', 'that', ',', 'in']
```

2.3 Converting tokens into token IDs

Next, let's convert these tokens from a Python string to an integer representation to produce the token IDs. This conversion is an intermediate step before converting the token IDs into embedding vectors.

To map the previously generated tokens into token IDs, we have to build a vocabulary first. This vocabulary defines how we map each unique word and special character to a unique integer, as shown in figure 2.6.

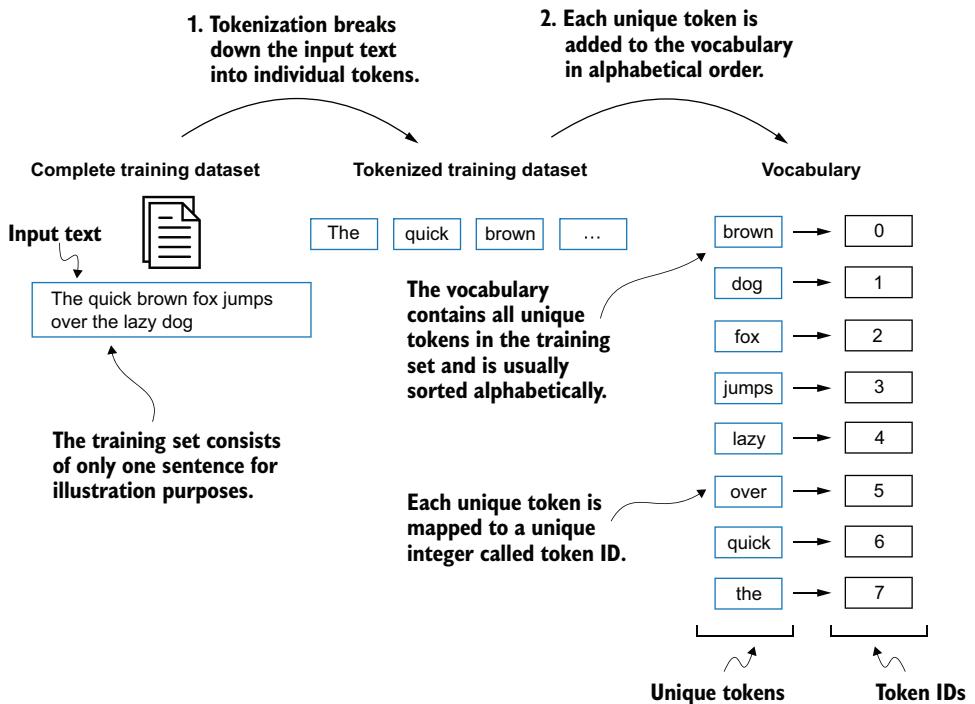


Figure 2.6 We build a vocabulary by tokenizing the entire text in a training dataset into individual tokens. These individual tokens are then sorted alphabetically, and duplicate tokens are removed. The unique tokens are then aggregated into a vocabulary that defines a mapping from each unique token to a unique integer value. The depicted vocabulary is purposely small and contains no punctuation or special characters for simplicity.

Now that we have tokenized Edith Wharton’s short story and assigned it to a Python variable called `preprocessed`, let’s create a list of all unique tokens and sort them alphabetically to determine the vocabulary size:

```
all_words = sorted(set(preprocessed))
vocab_size = len(all_words)
print(vocab_size)
```

After determining that the vocabulary size is 1,130 via this code, we create the vocabulary and print its first 51 entries for illustration purposes.

Listing 2.2 Creating a vocabulary

```
vocab = {token:integer for integer,token in enumerate(all_words)}
for i, item in enumerate(vocab.items()):
    print(item)
    if i >= 50:
        break
```

The output is

```
('!', 0)
('\'', 1)
("''", 2)
...
('Her', 49)
('Hermia', 50)
```

As we can see, the dictionary contains individual tokens associated with unique integer labels. Our next goal is to apply this vocabulary to convert new text into token IDs (figure 2.7).

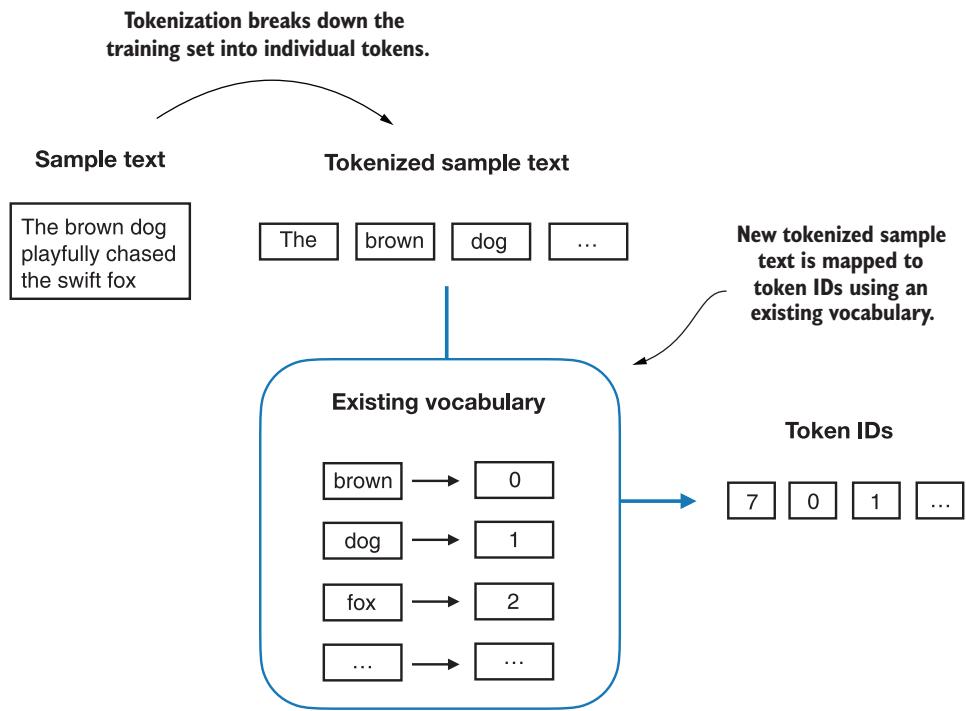


Figure 2.7 Starting with a new text sample, we tokenize the text and use the vocabulary to convert the text tokens into token IDs. The vocabulary is built from the entire training set and can be applied to the training set itself and any new text samples. The depicted vocabulary contains no punctuation or special characters for simplicity.

When we want to convert the outputs of an LLM from numbers back into text, we need a way to turn token IDs into text. For this, we can create an inverse version of the vocabulary that maps token IDs back to the corresponding text tokens.