Implementing Tokenization

March 31, 2025

1 Implementing Tokenization

Estimated time needed: **60** minutes

Tokenizers are essential tools in natural language processing that break down text into smaller units called tokens. These tokens can be words, characters, or subwords, making complex text understandable to computers. By dividing text into manageable pieces, tokenizers enable machines to process and analyze human language, powering various language-related applications like translation, sentiment analysis, and chatbots. Essentially, tokenizers bridge the gap between human language and machine understanding.

2 Table of Contents


```
<ol>
        <li><a href="#Word-based-tokenizer">Word-based tokenizer</a></li>
        <li><a href="#Character-based-tokenizer">Character-based tokenizer</a></li>
        <li><a href="#Subword-based-tokenizer">Subword-based tokenizer</a></li>
            <ol>
                <li><a href="#WordPiece">WordPiece</a></li>
                <li><a href="#Unigram-and-SentencePiece">Unigram and SentencePiece</a></li>
            </ol>
    </ol>
<li>
    <a href="#Tokenization-with-PyTorch">Tokenization with PyTorch</a>
</li>
<li>
    <a href="#Token-indices">Token indices</a>
    <ol>
        <li><a href="#Out-of-vocabulary-(OOV)">Out-of-vocabulary (OOV)</a></li>
```

```
    </ol>
</li>
<li><a href="#Exercise:-Comparative-text-tokenization-and-performance-analysis">Exercise: Compa
```

---

3 Objectives

After completing this lab, you will be able to:

- Understand the concept of tokenization and its importance in natural language processing
- Identify and explain word-based, character-based, and subword-based tokenization methods.
- Apply tokenization strategies to preprocess raw textual data before using it in machine learning models.

---

4 Setup

For this lab, you will be using the following libraries:

- `nltk` or natural language toolkit, will be employed for data management tasks. It offers comprehensive tools and resources for processing natural language text, making it a valuable choice for tasks such as text preprocessing and analysis.
- `spaCy` is an open-source software library for advanced natural language processing in Python. spaCy is renowned for its speed and accuracy in processing large volumes of text data.
- `BertTokenizer` is part of the Hugging Face Transformers library, a popular library for working with state-of-the-art pre-trained language models. BertTokenizer is specifically designed for tokenizing text according to the BERT model's specifications.
- `XLNetTokenizer` is another component of the Hugging Face Transformers library. It is tailored for tokenizing text in alignment with the XLNet model's requirements.
- `torchtext` It is part of the PyTorch ecosystem, to handle various natural language processing tasks. It simplifies the process of working with text data and provides functionalities for data preprocessing, tokenization, vocabulary management, and batching.

4.0.1 Installing required libraries

The following required libraries are **not** pre-installed in the Skills Network Labs environment. **You will need to run the following cell** to install them:

[1]:
```
!pip install nltk
!pip install transformers==4.42.1
!pip install sentencepiece
!pip install spacy
!python -m spacy download en_core_web_sm
!python -m spacy download de_core_news_sm
!pip install numpy scikit-learn
```

```
!pip install torch==2.2.2
!pip install torchtext==0.17.2
```

```
Requirement already satisfied: nltk in /opt/conda/lib/python3.12/site-packages
(3.9.1)
Requirement already satisfied: click in /opt/conda/lib/python3.12/site-packages
(from nltk) (8.1.8)
Requirement already satisfied: joblib in /opt/conda/lib/python3.12/site-packages
(from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/opt/conda/lib/python3.12/site-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.12/site-packages
(from nltk) (4.67.1)
Requirement already satisfied: transformers==4.42.1 in
/opt/conda/lib/python3.12/site-packages (4.42.1)
Requirement already satisfied: filelock in /opt/conda/lib/python3.12/site-
packages (from transformers==4.42.1) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (0.29.3)
Requirement already satisfied: numpy<2.0,>=1.17 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (1.26.0)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /opt/conda/lib/python3.12/site-
packages (from transformers==4.42.1) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (2024.11.6)
Requirement already satisfied: requests in /opt/conda/lib/python3.12/site-
packages (from transformers==4.42.1) (2.32.3)
Requirement already satisfied: tokenizers<0.20,>=0.19 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (0.19.1)
Requirement already satisfied: safetensors>=0.4.1 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /opt/conda/lib/python3.12/site-
packages (from transformers==4.42.1) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in
/opt/conda/lib/python3.12/site-packages (from huggingface-
hub<1.0,>=0.23.2->transformers==4.42.1) (2025.3.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/opt/conda/lib/python3.12/site-packages (from huggingface-
hub<1.0,>=0.23.2->transformers==4.42.1) (4.12.2)
Requirement already satisfied: charset_normalizer<4,>=2 in
/opt/conda/lib/python3.12/site-packages (from requests->transformers==4.42.1)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.12/site-
packages (from requests->transformers==4.42.1) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.12/site-packages (from requests->transformers==4.42.1)
```

```
(2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.12/site-packages (from requests->transformers==4.42.1)
(2024.12.14)
Requirement already satisfied: sentencepiece in /opt/conda/lib/python3.12/site-
packages (0.2.0)
Collecting spacy
  Downloading
spacy-3.8.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(27 kB)
Collecting spacy-legacy<3.1.0,>=3.0.11 (from spacy)
  Downloading spacy_legacy-3.0.12-py2.py3-none-any.whl.metadata (2.8 kB)
Collecting spacy-loggers<2.0.0,>=1.0.0 (from spacy)
  Downloading spacy_loggers-1.0.5-py3-none-any.whl.metadata (23 kB)
Collecting murmurhash<1.1.0,>=0.28.0 (from spacy)
  Downloading murmurhash-1.0.12-cp312-cp312-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (2.1 kB)
Collecting cymem<2.1.0,>=2.0.2 (from spacy)
  Downloading
cymem-2.0.11-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(8.5 kB)
Collecting preshed<3.1.0,>=3.0.2 (from spacy)
  Downloading preshed-3.0.9-cp312-cp312-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (2.2 kB)
Collecting thinc<8.4.0,>=8.3.4 (from spacy)
  Downloading
thinc-8.3.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(15 kB)
Collecting wasabi<1.2.0,>=0.9.1 (from spacy)
  Downloading wasabi-1.1.3-py3-none-any.whl.metadata (28 kB)
Collecting srsly<3.0.0,>=2.4.3 (from spacy)
  Downloading
srsly-2.5.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(19 kB)
Collecting catalogue<2.1.0,>=2.0.6 (from spacy)
  Downloading catalogue-2.0.10-py3-none-any.whl.metadata (14 kB)
Collecting weasel<0.5.0,>=0.1.0 (from spacy)
  Downloading weasel-0.4.1-py3-none-any.whl.metadata (4.6 kB)
Collecting typer<1.0.0,>=0.3.0 (from spacy)
  Downloading typer-0.15.2-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in
/opt/conda/lib/python3.12/site-packages (from spacy) (4.67.1)
Requirement already satisfied: numpy>=1.19.0 in /opt/conda/lib/python3.12/site-
packages (from spacy) (1.26.0)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in
/opt/conda/lib/python3.12/site-packages (from spacy) (2.32.3)
```

```
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in
/opt/conda/lib/python3.12/site-packages (from spacy) (2.10.6)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.12/site-packages
(from spacy) (3.1.5)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.12/site-
packages (from spacy) (75.8.0)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from spacy) (24.2)
Collecting langcodes<4.0.0,>=3.2.0 (from spacy)
  Downloading langcodes-3.5.0-py3-none-any.whl.metadata (29 kB)
Collecting language-data>=1.2 (from langcodes<4.0.0,>=3.2.0->spacy)
  Downloading language_data-1.3.0-py3-none-any.whl.metadata (4.3 kB)
Requirement already satisfied: annotated-types>=0.6.0 in
/opt/conda/lib/python3.12/site-packages (from
pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (0.7.0)
Requirement already satisfied: pydantic-core==2.27.2 in
/opt/conda/lib/python3.12/site-packages (from
pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (2.27.2)
Requirement already satisfied: typing-extensions>=4.12.2 in
/opt/conda/lib/python3.12/site-packages (from
pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (4.12.2)
Requirement already satisfied: charset_normalizer<4,>=2 in
/opt/conda/lib/python3.12/site-packages (from requests<3.0.0,>=2.13.0->spacy)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.12/site-
packages (from requests<3.0.0,>=2.13.0->spacy) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.12/site-packages (from requests<3.0.0,>=2.13.0->spacy)
(2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.12/site-packages (from requests<3.0.0,>=2.13.0->spacy)
(2024.12.14)
Collecting blis<1.3.0,>=1.2.0 (from thinc<8.4.0,>=8.3.4->spacy)
  Downloading
blis-1.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(7.7 kB)
Collecting confection<1.0.0,>=0.0.1 (from thinc<8.4.0,>=8.3.4->spacy)
  Downloading confection-0.1.5-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: click>=8.0.0 in /opt/conda/lib/python3.12/site-
packages (from typer<1.0.0,>=0.3.0->spacy) (8.1.8)
Collecting shellingham>=1.3.0 (from typer<1.0.0,>=0.3.0->spacy)
  Downloading shellingham-1.5.4-py2.py3-none-any.whl.metadata (3.5 kB)
Requirement already satisfied: rich>=10.11.0 in /opt/conda/lib/python3.12/site-
packages (from typer<1.0.0,>=0.3.0->spacy) (14.0.0)
Collecting cloudpathlib<1.0.0,>=0.7.0 (from weasel<0.5.0,>=0.1.0->spacy)
  Downloading cloudpathlib-0.21.0-py3-none-any.whl.metadata (14 kB)
Collecting smart-open<8.0.0,>=5.2.1 (from weasel<0.5.0,>=0.1.0->spacy)
  Downloading smart_open-7.1.0-py3-none-any.whl.metadata (24 kB)
```

```
Requirement already satisfied: MarkupSafe>=2.0 in
/opt/conda/lib/python3.12/site-packages (from jinja2->spacy) (3.0.2)
Collecting marisa-trie>=1.1.0 (from language-
data>=1.2->langcodes<4.0.0,>=3.2.0->spacy)
  Downloading marisa_trie-1.2.1-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.0 kB)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/opt/conda/lib/python3.12/site-packages (from
rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/opt/conda/lib/python3.12/site-packages (from
rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy) (2.19.1)
Requirement already satisfied: wrapt in /opt/conda/lib/python3.12/site-packages
(from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.1.0->spacy) (1.17.2)
Requirement already satisfied: mdurl~=0.1 in /opt/conda/lib/python3.12/site-
packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy)
(0.1.2)
Downloading
spacy-3.8.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (31.8 MB)
                         31.8/31.8 MB
182.9 MB/s eta 0:00:00
Downloading catalogue-2.0.10-py3-none-any.whl (17 kB)
Downloading
cymem-2.0.11-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (227 kB)
Downloading langcodes-3.5.0-py3-none-any.whl (182 kB)
Downloading murmurhash-1.0.12-cp312-cp312-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (138 kB)
Downloading preshed-3.0.9-cp312-cp312-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (156 kB)
Downloading spacy_legacy-3.0.12-py2.py3-none-any.whl (29 kB)
Downloading spacy_loggers-1.0.5-py3-none-any.whl (22 kB)
Downloading
srsly-2.5.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1 MB)
                         1.1/1.1 MB
80.5 MB/s eta 0:00:00
Downloading
thinc-8.3.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.7 MB)
                         3.7/3.7 MB
131.4 MB/s eta 0:00:00
Downloading typer-0.15.2-py3-none-any.whl (45 kB)
Downloading wasabi-1.1.3-py3-none-any.whl (27 kB)
Downloading weasel-0.4.1-py3-none-any.whl (50 kB)
Downloading
blis-1.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.6 MB)
                         11.6/11.6 MB
162.6 MB/s eta 0:00:00
```

```
Downloading cloudpathlib-0.21.0-py3-none-any.whl (52 kB)
Downloading confection-0.1.5-py3-none-any.whl (35 kB)
Downloading language_data-1.3.0-py3-none-any.whl (5.4 MB)
                         5.4/5.4 MB
159.1 MB/s eta 0:00:00
Downloading shellingham-1.5.4-py2.py3-none-any.whl (9.8 kB)
Downloading smart_open-7.1.0-py3-none-any.whl (61 kB)
Downloading
marisa_trie-1.2.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(1.4 MB)
                         1.4/1.4 MB
86.2 MB/s eta 0:00:00
Installing collected packages: cymem, wasabi, spacy-loggers, spacy-legacy,
smart-open, shellingham, murmurhash, marisa-trie, cloudpathlib, catalogue, blis,
srsly, preshed, language-data, typer, langcodes, confection, weasel, thinc,
spacy
Successfully installed blis-1.2.0 catalogue-2.0.10 cloudpathlib-0.21.0
confection-0.1.5 cymem-2.0.11 langcodes-3.5.0 language-data-1.3.0 marisa-
trie-1.2.1 murmurhash-1.0.12 preshed-3.0.9 shellingham-1.5.4 smart-open-7.1.0
spacy-3.8.4 spacy-legacy-3.0.12 spacy-loggers-1.0.5 srsly-2.5.1 thinc-8.3.4
typer-0.15.2 wasabi-1.1.3 weasel-0.4.1
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-
models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-
any.whl (12.8 MB)
                         12.8/12.8 MB
138.8 MB/s eta 0:00:00
Installing collected packages: en-core-web-sm
Successfully installed en-core-web-sm-3.8.0
 Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
Collecting de-core-news-sm==3.8.0
  Downloading https://github.com/explosion/spacy-
models/releases/download/de_core_news_sm-3.8.0/de_core_news_sm-3.8.0-py3-none-
any.whl (14.6 MB)
                         14.6/14.6 MB
177.9 MB/s eta 0:00:00
Installing collected packages: de-core-news-sm
Successfully installed de-core-news-sm-3.8.0
 Download and installation successful
You can now load the package via spacy.load('de_core_news_sm')
Requirement already satisfied: numpy in /opt/conda/lib/python3.12/site-packages
(1.26.0)
Collecting scikit-learn
  Downloading scikit_learn-1.6.1-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Collecting scipy>=1.6.0 (from scikit-learn)
  Downloading
```

```
scipy-1.15.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(61 kB)
Requirement already satisfied: joblib>=1.2.0 in /opt/conda/lib/python3.12/site-
packages (from scikit-learn) (1.4.2)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading
scikit_learn-1.6.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(13.1 MB)
                         13.1/13.1 MB
133.3 MB/s eta 0:00:00
Downloading
scipy-1.15.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (37.3
MB)
                         37.3/37.3 MB
174.1 MB/s eta 0:00:0000:01
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, scikit-learn
Successfully installed scikit-learn-1.6.1 scipy-1.15.2 threadpoolctl-3.6.0
Requirement already satisfied: torch==2.2.2 in /opt/conda/lib/python3.12/site-
packages (2.2.2)
Requirement already satisfied: filelock in /opt/conda/lib/python3.12/site-
packages (from torch==2.2.2) (3.18.0)
Requirement already satisfied: typing-extensions>=4.8.0 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (4.12.2)
Requirement already satisfied: sympy in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2) (1.13.3)
Requirement already satisfied: networkx in /opt/conda/lib/python3.12/site-
packages (from torch==2.2.2) (3.4.2)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2) (3.1.5)
Requirement already satisfied: fsspec in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2) (2025.3.1)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in
```

```
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.19.3 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (2.19.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.105)
Requirement already satisfied: nvidia-nvjitlink-cu12 in
/opt/conda/lib/python3.12/site-packages (from nvidia-cusolver-
cu12==11.4.5.107->torch==2.2.2) (12.8.93)
Requirement already satisfied: MarkupSafe>=2.0 in
/opt/conda/lib/python3.12/site-packages (from jinja2->torch==2.2.2) (3.0.2)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/opt/conda/lib/python3.12/site-packages (from sympy->torch==2.2.2) (1.3.0)
Requirement already satisfied: torchtext==0.17.2 in
/opt/conda/lib/python3.12/site-packages (0.17.2)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.12/site-packages
(from torchtext==0.17.2) (4.67.1)
Requirement already satisfied: requests in /opt/conda/lib/python3.12/site-
packages (from torchtext==0.17.2) (2.32.3)
Requirement already satisfied: torch==2.2.2 in /opt/conda/lib/python3.12/site-
packages (from torchtext==0.17.2) (2.2.2)
Requirement already satisfied: numpy in /opt/conda/lib/python3.12/site-packages
(from torchtext==0.17.2) (1.26.0)
Requirement already satisfied: filelock in /opt/conda/lib/python3.12/site-
packages (from torch==2.2.2->torchtext==0.17.2) (3.18.0)
Requirement already satisfied: typing-extensions>=4.8.0 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(4.12.2)
Requirement already satisfied: sympy in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2->torchtext==0.17.2) (1.13.3)
Requirement already satisfied: networkx in /opt/conda/lib/python3.12/site-
packages (from torch==2.2.2->torchtext==0.17.2) (3.4.2)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2->torchtext==0.17.2) (3.1.5)
Requirement already satisfied: fsspec in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2->torchtext==0.17.2) (2025.3.1)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
```

```
(8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.19.3 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(2.19.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.105)
Requirement already satisfied: nvidia-nvjitlink-cu12 in
/opt/conda/lib/python3.12/site-packages (from nvidia-cusolver-
cu12==11.4.5.107->torch==2.2.2->torchtext==0.17.2) (12.8.93)
Requirement already satisfied: charset_normalizer<4,>=2 in
/opt/conda/lib/python3.12/site-packages (from requests->torchtext==0.17.2)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.12/site-
packages (from requests->torchtext==0.17.2) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.12/site-packages (from requests->torchtext==0.17.2)
(2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.12/site-packages (from requests->torchtext==0.17.2)
(2024.12.14)
Requirement already satisfied: MarkupSafe>=2.0 in
/opt/conda/lib/python3.12/site-packages (from
jinja2->torch==2.2.2->torchtext==0.17.2) (3.0.2)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/opt/conda/lib/python3.12/site-packages (from
sympy->torch==2.2.2->torchtext==0.17.2) (1.3.0)
```

4.0.2 Importing required libraries

We recommend you import all required libraries in one place (here):

[2]:
```
import nltk
nltk.download("punkt")
nltk.download('punkt_tab')
import spacy
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
from nltk.util import ngrams
from transformers import BertTokenizer
from transformers import XLNetTokenizer

from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator

def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')
```

```
[nltk_data] Downloading package punkt to /home/jupyterlab/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to
[nltk_data]     /home/jupyterlab/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

4.1 What is a tokenizer and why do we use it?

Tokenizers play a pivotal role in natural language processing, segmenting text into smaller units known as tokens. These tokens are subsequently transformed into numerical representations called token indices, which are directly employed by deep learning algorithms.

4.2 Types of tokenizer

The meaningful representation can vary depending on the model in use. Various models employ distinct tokenization algorithms, and you will broadly cover the following approaches. Transforming text into numerical values might appear straightforward initially, but it encompasses several considerations that must be kept in mind.

4.3 Word-based tokenizer

4.3.1 nltk

As the name suggests, this is the splitting of text based on words. There are different rules for word-based tokenizers, such as splitting on spaces or splitting on punctuation. Each option assigns a specific ID to the split word. Here you use nltk's `word_tokenize`

[3]:
```
text = "This is a sample sentence for word tokenization."
tokens = word_tokenize(text)
print(tokens)
```

```
['This', 'is', 'a', 'sample', 'sentence', 'for', 'word', 'tokenization', '.']
```

General libraries like nltk and spaCy often split words like 'don't' and 'couldn't,' which are contractions, into different individual words. There's no universal rule, and each library has its own tokenization rules for word-based tokenizers. However, the general guideline is to preserve the input format after tokenization to match how the model was trained.

[4]:
```
# This showcases word_tokenize from nltk library

text = "I couldn't help the dog. Can't you do it? Don't be afraid if you are."
tokens = word_tokenize(text)
print(tokens)
```

```
['I', 'could', "n't", 'help', 'the', 'dog', '.', 'Ca', "n't", 'you', 'do', 'it',
'?', 'Do', "n't", 'be', 'afraid', 'if', 'you', 'are', '.']
```

[5]:
```
# This showcases the use of the 'spaCy' tokenizer with torchtext's␣
 ↪get_tokenizer function

text = "I couldn't help the dog. Can't you do it? Don't be afraid if you are."
nlp = spacy.load("en_core_web_sm")
doc = nlp(text)

# Making a list of the tokens and priting the list
token_list = [token.text for token in doc]
print("Tokens:", token_list)

# Showing token details
for token in doc:
    print(token.text, token.pos_, token.dep_)
```

```
Tokens: ['I', 'could', "n't", 'help', 'the', 'dog', '.', 'Ca', "n't", 'you',
'do', 'it', '?', 'Do', "n't", 'be', 'afraid', 'if', 'you', 'are', '.']
I PRON nsubj
could AUX aux
n't PART neg
help VERB ROOT
the DET det
dog NOUN dobj
. PUNCT punct
Ca AUX aux
n't PART neg
you PRON nsubj
do VERB ROOT
it PRON dobj
```

```
? PUNCT punct
Do AUX aux
n't PART neg
be AUX ROOT
afraid ADJ acomp
if SCONJ mark
you PRON nsubj
are AUX advcl
. PUNCT punct
```

Explanation of a few lines: - I PRON nsubj: "I" is a pronoun (PRON) and is the nominal subject (nsubj) of the sentence. - help VERB ROOT: "help" is a verb (VERB) and is the root action (ROOT) of the sentence. - afraid ADJ acomp: "afraid" is an adjective (ADJ) and is an adjectival complement (acomp) which gives more information about a state or quality related to the verb.

The problem with this algorithm is that words with similar meanings will be assigned different IDs, resulting in them being treated as entirely separate words with distinct meanings. For example, *Unicorns* is the plural form of *Unicorn*, but a word-based tokenizer would tokenize them as two separate words, potentially causing the model to miss their semantic relationship.

[6]:
```
text = "Unicorns are real. I saw a unicorn yesterday."
token = word_tokenize(text)
print(token)
```

```
['Unicorns', 'are', 'real', '.', 'I', 'saw', 'a', 'unicorn', 'yesterday', '.']
```

Each word is split into a token, leading to a significant increase in the model's overall vocabulary. Each token is mapped to a large vector containing the word's meanings, resulting in large model parameters.

Languages generally have a large number of words, the vocabularies based on them will always be extensive. However, the number of characters in a language is always fewer compared to the number of words. Next, we will explore character-based tokenizers.

4.4 Character-based tokenizer

As the name suggests, character-based tokenization involves splitting text into individual characters. The advantage of using this approach is that the resulting vocabularies are inherently small. Furthermore, since languages have a limited set of characters, the number of out-of-vocabulary tokens is also limited, reducing token wastage.

For example: Input text: `This is a sample sentence for tokenization.`

Character-based tokenization output: `['T', 'h', 'i', 's', 'i', 's', 'a', 's', 'a', 'm', 'p', 'l', 'e', 's', 'e', 'n', 't', 'e', 'n', 'c', 'e', 'f', 'o', 'r', 't', 'o', 'k', 'e', 'n', 'i', 'z', 'a', 't', 'i', 'o', 'n', '.']`

However, it's important to note that character-based tokenization has its limitations. Single characters may not convey the same information as entire words, and the overall token length increases significantly, potentially causing issues with model size and a loss of performance.

You have explored the limitations of both word-based and character-based tokenization methods.

To leverage the advantages of both approaches, transformers employ subword-based tokenization, which will be discussed next.

4.5 Subword-based tokenizer

The subword-based tokenizer allows frequently used words to remain unsplit while breaking down infrequent words into meaningful subwords. Techniques such as SentencePiece, or WordPiece are commonly used for subword tokenization. These methods learn subword units from a given text corpus, identifying common prefixes, suffixes, and root words as subword tokens based on their frequency of occurrence. This approach offers the advantage of representing a broader range of words and adapting to the specific language patterns within a text corpus.

In both examples below, words are split into subwords, which helps preserve the semantic information associated with the overall word. For instance, 'Unhappiness' is split into 'un' and 'happiness,' both of which can appear as stand-alone subwords. When we combine these individual subwords, they form 'unhappiness,' which retains its meaningful context. This approach aids in maintaining the overall information and semantic meaning of words.

4.5.1 WordPiece

Initially, WordPiece initializes its vocabulary to include every character present in the training data and progressively learns a specified number of merge rules. WordPiece doesn't select the most frequent symbol pair but rather the one that maximizes the likelihood of the training data when added to the vocabulary. In essence, WordPiece evaluates what it sacrifices by merging two symbols to ensure it's a worthwhile endeavor.

Now, the WordPiece tokenizer is implemented in BertTokenizer. Note that BertTokenizer treats composite words as separate tokens.

[7]:
```
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
tokenizer.tokenize("IBM taught me tokenization.")
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
```

[7]:
```
['ibm', 'taught', 'me', 'token', '##ization', '.']
```

Here's a breakdown of the output: - 'ibm': "IBM" is tokenized as 'ibm'. BERT converts tokens into lowercase, as it does not retain the case information when using the "bert-base-uncased" model. - 'taught', 'me', '.': These tokens are the same as the original words or punctuation, just lowercased (except punctuation). - 'token', '##ization': "Tokenization" is broken into two tokens. "Token" is a whole word, and "##ization" is a part of the original word. The "##" indicates that "ization" should be connected back to "token" when detokenizing (transforming tokens back to words).

4.5.2 Unigram and SentencePiece

Unigram is a method for breaking words or text into smaller pieces. It accomplishes this by starting with a large list of possibilities and gradually narrowing it down based on how frequently those pieces appear in the text. This approach aids in efficient text tokenization.

SentencePiece is a tool that takes text, divides it into smaller, more manageable parts, assigns IDs to these segments, and ensures that it does so consistently. Consequently, if you use SentencePiece on the same text repeatedly, you will consistently obtain the same subwords and IDs.

Unigram and SentencePiece work together by implementing Unigram's subword tokenization method within the SentencePiece framework. SentencePiece handles subword segmentation and ID assignment, while Unigram's principles guide the vocabulary reduction process to create a more efficient representation of the text data. This combination is particularly valuable for various NLP tasks in which subword tokenization can enhance the performance of language models.

[8]:
```
tokenizer = XLNetTokenizer.from_pretrained("xlnet-base-cased")
tokenizer.tokenize("IBM taught me tokenization.")
```

```
spiece.model:   0%|          | 0.00/798k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/1.38M [00:00<?, ?B/s]

config.json:   0%|          | 0.00/760 [00:00<?, ?B/s]
```

[8]:
```
[' IBM', ' taught', ' me', ' token', 'ization', '.']
```

Here's what's happening with each token: - ' IBM': The " " (often referred to as "whitespace character") before "IBM" indicates that this token is preceded by a space in the original text. "IBM" is kept as is because it's recognized as a whole token by XLNet and it preserves the casing because you are using the "xlnet-base-cased" model. - ' taught', ' me', ' token': Similarly, these tokens are prefixed with " " to indicate they are new words preceded by a space in the original text, preserving the word as a whole and maintaining the original casing. - 'ization': Unlike "BertTokenizer," "XLNetTokenizer" does not use "##" to indicate subword tokens. "ization" appears as its own token without a prefix because it directly follows the preceding word "token" without a space in the original text. - '.': The period is tokenized as a separate token since punctuation is treated separately.

4.6 Tokenization with PyTorch

In PyTorch, especially with the `torchtext` library, the tokenizer breaks down text from a data set into individual words or subwords, facilitating their conversion into numerical format. After tokenization, the vocab (vocabulary) maps these tokens to unique integers, allowing them to be fed into neural networks. This process is vital because deep learning models operate on numerical data and cannot process raw text directly. Thus, tokenization and vocabulary mapping serve as a bridge between human-readable text and machine-operable numerical data. Consider the dataset:

[9]:
```
dataset = [
    (1,"Introduction to NLP"),
    (2,"Basics of PyTorch"),
    (1,"NLP Techniques for Text Classification"),
```

```
    (3,"Named Entity Recognition with PyTorch"),
    (3,"Sentiment Analysis using PyTorch"),
    (3,"Machine Translation with PyTorch"),
    (1," NLP Named Entity,Sentiment Analysis,Machine Translation "),
    (1," Machine Translation with NLP "),
    (1," Named Entity vs Sentiment Analysis  NLP ")]
```

This next line imports the `get_tokenizer` function from the `torchtext.data.utils` module. In the torchtext library, the `get_tokenizer` function is utilized to fetch a tokenizer by name. It provides support for a range of tokenization methods, including basic string splitting, and returns various tokenizers based on the argument passed to it.

[10]:
```
from torchtext.data.utils import get_tokenizer
```

[11]:
```
tokenizer = get_tokenizer("basic_english")
```

You apply the tokenizer to the dataset. Note: If `basic_english` is selected, it returns the `_basic_english_normalize()` function, which normalizes the string first and then splits it by space.

[12]:
```
tokenizer(dataset[0][1])
```

[12]:
```
['introduction', 'to', 'nlp']
```

4.7 Token indices

You would represent words as numbers as NLP algorithms can process and manipulate numbers more efficiently and quickly than raw text. You use the function **`build_vocab_from_iterator`**, the output is typically referred to as 'token indices' or simply 'indices.' These indices represent the numeric representations of the tokens in the vocabulary.

The **`build_vocab_from_iterator`** function, when applied to a list of tokens, assigns a unique index to each token based on its position in the vocabulary. These indices serve as a way to represent the tokens in a numerical format that can be easily processed by machine learning models.

For example, given a vocabulary with tokens ["apple", "banana", "orange"], the corresponding indices might be [0, 1, 2], where "apple" is represented by index 0, "banana" by index 1, and "orange" by index 2.

`dataset` is an iterable. Therefore, you use a generator function yield_tokens to apply the **`tokenizer`**. The purpose of the generator function **`yield_tokens`** is to yield tokenized texts one at a time. Instead of processing the entire dataset and returning all the tokenized texts in one go, the generator function processes and yields each tokenized text individually as it is requested. The tokenization process is performed lazily, which means the next tokenized text is generated only when needed, saving memory and computational resources.

[13]:
```
def yield_tokens(data_iter):
    for  _,text in data_iter:
        yield tokenizer(text)
```

[14]:
```
my_iterator = yield_tokens(dataset)
```

This creates an iterator called **my_iterator** using the generator. To begin the evaluation of the generator and retrieve the values, you can iterate over **my_iterator** using a for loop or retrieve values from it using the **next()** function.

[15]:
```
next(my_iterator)
```

[15]:
```
['introduction', 'to', 'nlp']
```

You build a vocabulary from the tokenized texts generated by the **yield_tokens** generator function, which processes the dataset. The **build_vocab_from_iterator()** function constructs the vocabulary, including a special token `unk` to represent out-of-vocabulary words.

4.7.1 Out-of-vocabulary (OOV)

When text data is tokenized, there may be words that are not present in the vocabulary because they are rare or unseen during the vocabulary building process. When encountering such OOV words during actual language processing tasks like text generation or language modeling, the model can use the `<unk>` token to represent them.

For example, if the word "apple" is present in the vocabulary, but "pineapple" is not, "apple" will be used normally in the text, but "pineapple" (being an OOV word) would be replaced by the `<unk>` token.

By including the `<unk>` token in the vocabulary, you provide a consistent way to handle out-of-vocabulary words in your language model or other natural language processing tasks.

[16]:
```
vocab = build_vocab_from_iterator(yield_tokens(dataset), specials=["<unk>"])
vocab.set_default_index(vocab["<unk>"])
```

This code demonstrates how to fetch a tokenized sentence from an iterator, convert its tokens into indices using a provided vocabulary, and then print both the original sentence and its corresponding indices.

[17]:
```
def get_tokenized_sentence_and_indices(iterator):
    tokenized_sentence = next(iterator)  # Get the next tokenized sentence
    token_indices = [vocab[token] for token in tokenized_sentence]  # Get token␣
 ↪indices
    return tokenized_sentence, token_indices

tokenized_sentence, token_indices =␣
 ↪get_tokenized_sentence_and_indices(my_iterator)
next(my_iterator)

print("Tokenized Sentence:", tokenized_sentence)
print("Token Indices:", token_indices)
```

```
Tokenized Sentence: ['basics', 'of', 'pytorch']
Token Indices: [11, 15, 2]
```

Using the lines of code provided above in a simple example, demonstrate tokenization and the building of vocabulary in PyTorch.

[18]:
```
lines = ["IBM taught me tokenization",
         "Special tokenizers are ready and they will blow your mind",
         "just saying hi!"]

special_symbols = ['<unk>', '<pad>', '<bos>', '<eos>']

tokenizer_en = get_tokenizer('spacy', language='en_core_web_sm')

tokens = []
max_length = 0

for line in lines:
    tokenized_line = tokenizer_en(line)
    tokenized_line = ['<bos>'] + tokenized_line + ['<eos>']
    tokens.append(tokenized_line)
    max_length = max(max_length, len(tokenized_line))

for i in range(len(tokens)):
    tokens[i] = tokens[i] + ['<pad>'] * (max_length - len(tokens[i]))

print("Lines after adding special tokens:\n", tokens)

# Build vocabulary without unk_init
vocab = build_vocab_from_iterator(tokens, specials=['<unk>'])
vocab.set_default_index(vocab["<unk>"])

# Vocabulary and Token Ids
print("Vocabulary:", vocab.get_itos())
print("Token IDs for 'tokenization':", vocab.get_stoi())
```

```
Lines after adding special tokens:
 [['<bos>', 'IBM', 'taught', 'me', 'tokenization', '<eos>', '<pad>', '<pad>',
'<pad>', '<pad>', '<pad>', '<pad>'], ['<bos>', 'Special', 'tokenizers', 'are',
'ready', 'and', 'they', 'will', 'blow', 'your', 'mind', '<eos>'], ['<bos>',
'just', 'saying', 'hi', '!', '<eos>', '<pad>', '<pad>', '<pad>', '<pad>',
'<pad>', '<pad>']]
Vocabulary: ['<unk>', '<pad>', '<bos>', '<eos>', '!', 'IBM', 'Special', 'and',
'are', 'blow', 'hi', 'just', 'me', 'mind', 'ready', 'saying', 'taught', 'they',
'tokenization', 'tokenizers', 'will', 'your']
Token IDs for 'tokenization': {'will': 20, 'tokenizers': 19, 'tokenization': 18,
'taught': 16, 'your': 21, 'saying': 15, '<unk>': 0, 'and': 7, 'hi': 10, '<pad>':
1, '<bos>': 2, 'they': 17, '<eos>': 3, '!': 4, 'ready': 14, 'IBM': 5, 'are': 8,
'Special': 6, 'mind': 13, 'me': 12, 'blow': 9, 'just': 11}
```

Let's break down the output: 1. **Special Tokens**: - Token: "<unk>", Index: 0: <unk> stands for "unknown" and represents words that were not seen during vocabulary building, usually during

inference on new text. - Token: "<pad>", Index: 1: <pad> is a "padding" token used to make sequences of words the same length when batching them together. - Token: "<bos>", Index: 2: <bos> is an acronym for "beginning of sequence" and is used to denote the start of a text sequence. - Token: "<eos>", Index: 3: <eos> is an acronym for "end of sequence" and is used to denote the end of a text sequence.

2. **Word Tokens**: The rest of the tokens are words or punctuation extracted from the provided sentences, each assigned a unique index:

- Token: "IBM", Index: 5
- Token: "taught", Index: 16
- Token: "me", Index: 12 … and so on.

3. **Vocabulary**: It denotes the total number of tokens in the sentences upon which vocabulary is built.

4. **Token IDs for 'tokenization'**: It represents the token IDs assigned in the vocab where a number represents its presence in the sentence.

[19]:
```
new_line = "I learned about embeddings and attention mechanisms."

# Tokenize the new line
tokenized_new_line = tokenizer_en(new_line)
tokenized_new_line = ['<bos>'] + tokenized_new_line + ['<eos>']

# Pad the new line to match the maximum length of previous lines
new_line_padded = tokenized_new_line + ['<pad>'] * (max_length -␣
 ↪len(tokenized_new_line))

# Convert tokens to IDs and handle unknown words
new_line_ids = [vocab[token] if token in vocab else vocab['<unk>'] for token in␣
 ↪new_line_padded]

# Example usage
print("Token IDs for new line:", new_line_ids)
```

```
Token IDs for new line: [2, 0, 0, 0, 0, 7, 0, 0, 0, 3, 1, 1]
```

Let's break down the output:

1. **Special Tokens**:

- Token: "<unk>", Index: 0: <unk> stands for "unknown" and represents words that were not seen during vocabulary building, usually during inference on new text.
- Token: "<pad>", Index: 1: <pad> is a "padding" token used to make sequences of words the same length when batching them together.
- Token: "<bos>", Index: 2: <bos> is an acronym for "beginning of sequence" and is used to denote the start of a text sequence.
- Token: "<eos>", Index: 3: <eos> is an acronym for "end of sequence" and is used to denote the end of a text sequence.

2. The token **and** is recognized in the sentence and it is assigned **token_id - 7**.

4.8 Exercise: Comparative text tokenization and performance analysis

- Objective: Evaluate and compare the tokenization capabilities of four different NLP libraries (`nltk`, `spaCy`, `BertTokenizer`, and `XLNetTokenizer`) by analyzing the frequency of tokenized words and measuring the processing time for each tool using `datetime`.
- Text for tokenization is as below:

[20]:
```
text = """
Going through the world of tokenization has been like walking through a huge␣
 ↪maze made of words, symbols, and meanings. Each turn shows a bit more about␣
 ↪the cool ways computers learn to understand our language. And while I'm␣
 ↪still finding my way through it, the journey's been enlightening and,␣
 ↪honestly, a bunch of fun.
Eager to see where this learning path takes me next!"
"""

# Counting and displaying tokens and their frequency
from collections import Counter
def show_frequencies(tokens, method_name):
    print(f"{method_name} Token Frequencies: {dict(Counter(tokens))}\n")
```

[22]:
```
# TODO
import nltk
import spacy
from transformers import BertTokenizer, XLNetTokenizer
from datetime import datetime

# NLTK Tokenization
start_time = datetime.now()
nltk_tokens = nltk.word_tokenize(text)
nltk_time = datetime.now() - start_time

# SpaCy Tokenization
nlp = spacy.load("en_core_web_sm")
start_time = datetime.now()
spacy_tokens = [token.text for token in nlp(text)]
spacy_time = datetime.now() - start_time

# BertTokenizer Tokenization
bert_tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
start_time = datetime.now()
bert_tokens = bert_tokenizer.tokenize(text)
bert_time = datetime.now() - start_time

# XLNetTokenizer Tokenization
xlnet_tokenizer = XLNetTokenizer.from_pretrained("xlnet-base-cased")
start_time = datetime.now()
xlnet_tokens = xlnet_tokenizer.tokenize(text)
```

```
xlnet_time = datetime.now() - start_time

# Display tokens, time taken for each tokenizer, and token frequencies
print(f"NLTK Tokens: {nltk_tokens}\nTime Taken: {nltk_time} seconds\n")
show_frequencies(nltk_tokens, "NLTK")

print(f"SpaCy Tokens: {spacy_tokens}\nTime Taken: {spacy_time} seconds\n")
show_frequencies(spacy_tokens, "SpaCy")

print(f"Bert Tokens: {bert_tokens}\nTime Taken: {bert_time} seconds\n")
show_frequencies(bert_tokens, "Bert")

print(f"XLNet Tokens: {xlnet_tokens}\nTime Taken: {xlnet_time} seconds\n")
show_frequencies(xlnet_tokens, "XLNet")
```

```
NLTK Tokens: ['Going', 'through', 'the', 'world', 'of', 'tokenization', 'has',
'been', 'like', 'walking', 'through', 'a', 'huge', 'maze', 'made', 'of',
'words', ',', 'symbols', ',', 'and', 'meanings', '.', 'Each', 'turn', 'shows',
'a', 'bit', 'more', 'about', 'the', 'cool', 'ways', 'computers', 'learn', 'to',
'understand', 'our', 'language', '.', 'And', 'while', 'I', "'m", 'still',
'finding', 'my', 'way', 'through', 'it', ',', 'the', 'journey', '’', 's',
'been', 'enlightening', 'and', ',', 'honestly', ',', 'a', 'bunch', 'of', 'fun',
'.', 'Eager', 'to', 'see', 'where', 'this', 'learning', 'path', 'takes', 'me',
'next', '!', "'"]
Time Taken: 0:00:00.000456 seconds

NLTK Token Frequencies: {'Going': 1, 'through': 3, 'the': 3, 'world': 1, 'of':
3, 'tokenization': 1, 'has': 1, 'been': 2, 'like': 1, 'walking': 1, 'a': 3,
'huge': 1, 'maze': 1, 'made': 1, 'words': 1, ',': 5, 'symbols': 1, 'and': 2,
'meanings': 1, '.': 3, 'Each': 1, 'turn': 1, 'shows': 1, 'bit': 1, 'more': 1,
'about': 1, 'cool': 1, 'ways': 1, 'computers': 1, 'learn': 1, 'to': 2,
'understand': 1, 'our': 1, 'language': 1, 'And': 1, 'while': 1, 'I': 1, "'m": 1,
'still': 1, 'finding': 1, 'my': 1, 'way': 1, 'it': 1, 'journey': 1, '’': 1, 's':
1, 'enlightening': 1, 'honestly': 1, 'bunch': 1, 'fun': 1, 'Eager': 1, 'see': 1,
'where': 1, 'this': 1, 'learning': 1, 'path': 1, 'takes': 1, 'me': 1, 'next': 1,
'!': 1, "'": 1}

SpaCy Tokens: ['\n', 'Going', 'through', 'the', 'world', 'of', 'tokenization',
'has', 'been', 'like', 'walking', 'through', 'a', 'huge', 'maze', 'made', 'of',
'words', ',', 'symbols', ',', 'and', 'meanings', '.', 'Each', 'turn', 'shows',
'a', 'bit', 'more', 'about', 'the', 'cool', 'ways', 'computers', 'learn', 'to',
'understand', 'our', 'language', '.', 'And', 'while', 'I', "'m", 'still',
'finding', 'my', 'way', 'through', 'it', ',', 'the', 'journey', '’s', 'been',
'enlightening', 'and', ',', 'honestly', ',', 'a', 'bunch', 'of', 'fun', '.',
'\n', 'Eager', 'to', 'see', 'where', 'this', 'learning', 'path', 'takes', 'me',
'next', '!', '"', '\n']
Time Taken: 0:00:00.021031 seconds
```

```
SpaCy Token Frequencies: {'\n': 3, 'Going': 1, 'through': 3, 'the': 3, 'world':
1, 'of': 3, 'tokenization': 1, 'has': 1, 'been': 2, 'like': 1, 'walking': 1,
'a': 3, 'huge': 1, 'maze': 1, 'made': 1, 'words': 1, ',': 5, 'symbols': 1,
'and': 2, 'meanings': 1, '.': 3, 'Each': 1, 'turn': 1, 'shows': 1, 'bit': 1,
'more': 1, 'about': 1, 'cool': 1, 'ways': 1, 'computers': 1, 'learn': 1, 'to':
2, 'understand': 1, 'our': 1, 'language': 1, 'And': 1, 'while': 1, 'I': 1, "'m":
1, 'still': 1, 'finding': 1, 'my': 1, 'way': 1, 'it': 1, 'journey': 1, '’s': 1,
'enlightening': 1, 'honestly': 1, 'bunch': 1, 'fun': 1, 'Eager': 1, 'see': 1,
'where': 1, 'this': 1, 'learning': 1, 'path': 1, 'takes': 1, 'me': 1, 'next': 1,
'!': 1, '"': 1}

Bert Tokens: ['going', 'through', 'the', 'world', 'of', 'token', '##ization',
'has', 'been', 'like', 'walking', 'through', 'a', 'huge', 'maze', 'made', 'of',
'words', ',', 'symbols', ',', 'and', 'meanings', '.', 'each', 'turn', 'shows',
'a', 'bit', 'more', 'about', 'the', 'cool', 'ways', 'computers', 'learn', 'to',
'understand', 'our', 'language', '.', 'and', 'while', 'i', "'", 'm', 'still',
'finding', 'my', 'way', 'through', 'it', ',', 'the', 'journey', '’', 's',
'been', 'en', '##light', '##ening', 'and', ',', 'honestly', ',', 'a', 'bunch',
'of', 'fun', '.', 'eager', 'to', 'see', 'where', 'this', 'learning', 'path',
'takes', 'me', 'next', '!', '"']
Time Taken: 0:00:00.001244 seconds

Bert Token Frequencies: {'going': 1, 'through': 3, 'the': 3, 'world': 1, 'of':
3, 'token': 1, '##ization': 1, 'has': 1, 'been': 2, 'like': 1, 'walking': 1,
'a': 3, 'huge': 1, 'maze': 1, 'made': 1, 'words': 1, ',': 5, 'symbols': 1,
'and': 3, 'meanings': 1, '.': 3, 'each': 1, 'turn': 1, 'shows': 1, 'bit': 1,
'more': 1, 'about': 1, 'cool': 1, 'ways': 1, 'computers': 1, 'learn': 1, 'to':
2, 'understand': 1, 'our': 1, 'language': 1, 'while': 1, 'i': 1, "'": 1, 'm': 1,
'still': 1, 'finding': 1, 'my': 1, 'way': 1, 'it': 1, 'journey': 1, '’': 1, 's':
1, 'en': 1, '##light': 1, '##ening': 1, 'honestly': 1, 'bunch': 1, 'fun': 1,
'eager': 1, 'see': 1, 'where': 1, 'this': 1, 'learning': 1, 'path': 1, 'takes':
1, 'me': 1, 'next': 1, '!': 1, '"': 1}

XLNet Tokens: [' Going', ' through', ' the', ' world', ' of', ' token',
'ization', ' has', ' been', ' like', ' walking', ' through', ' a', ' huge',
' maze', ' made', ' of', ' words', ',', ' symbols', ',', ' and', ' meaning',
's', '.', ' Each', ' turn', ' shows', ' a', ' bit', ' more', ' about', ' the',
' cool', ' ways', ' computers', ' learn', ' to', ' understand', ' our',
' language', '.', ' And', ' while', ' I', "'", 'm', ' still', ' finding', ' my',
' way', ' through', ' it', ',', ' the', ' journey', '’', 's', ' been',
' enlighten', 'ing', ' and', ',', ' honestly', ',', ' a', ' bunch', ' of',
' fun', '.', ' E', 'ager', ' to', ' see', ' where', ' this', ' learning',
' path', ' takes', ' me', ' next', '!', '"']
Time Taken: 0:00:00.000474 seconds

XLNet Token Frequencies: {' Going': 1, ' through': 3, ' the': 3, ' world': 1,
' of': 3, ' token': 1, 'ization': 1, ' has': 1, ' been': 2, ' like': 1,
' walking': 1, ' a': 3, ' huge': 1, ' maze': 1, ' made': 1, ' words': 1, ',': 5,
```

```
' symbols': 1, ' and': 2, ' meaning': 1, 's': 2, '.': 3, ' Each': 1, ' turn': 1,
' shows': 1, ' bit': 1, ' more': 1, ' about': 1, ' cool': 1, ' ways': 1,
' computers': 1, ' learn': 1, ' to': 2, ' understand': 1, ' our': 1,
' language': 1, ' And': 1, ' while': 1, ' I': 1, "'": 1, 'm': 1, ' still': 1,
' finding': 1, ' my': 1, ' way': 1, ' it': 1, ' journey': 1, '’': 1,
' enlighten': 1, 'ing': 1, ' honestly': 1, ' bunch': 1, ' fun': 1, ' E': 1,
'ager': 1, ' see': 1, ' where': 1, ' this': 1, ' learning': 1, ' path': 1,
' takes': 1, ' me': 1, ' next': 1, '!': 1, '"': 1}
```

Click here for the answer

```
import nltk
import spacy
from transformers import BertTokenizer, XLNetTokenizer
from datetime import datetime

# NLTK Tokenization
start_time = datetime.now()
nltk_tokens = nltk.word_tokenize(text)
nltk_time = datetime.now() - start_time

# SpaCy Tokenization
nlp = spacy.load("en_core_web_sm")
start_time = datetime.now()
spacy_tokens = [token.text for token in nlp(text)]
spacy_time = datetime.now() - start_time

# BertTokenizer Tokenization
bert_tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
start_time = datetime.now()
bert_tokens = bert_tokenizer.tokenize(text)
bert_time = datetime.now() - start_time

# XLNetTokenizer Tokenization
xlnet_tokenizer = XLNetTokenizer.from_pretrained("xlnet-base-cased")
start_time = datetime.now()
xlnet_tokens = xlnet_tokenizer.tokenize(text)
xlnet_time = datetime.now() - start_time

# Display tokens, time taken for each tokenizer, and token frequencies
print(f"NLTK Tokens: {nltk_tokens}\nTime Taken: {nltk_time} seconds\n")
show_frequencies(nltk_tokens, "NLTK")

print(f"SpaCy Tokens: {spacy_tokens}\nTime Taken: {spacy_time} seconds\n")
show_frequencies(spacy_tokens, "SpaCy")

print(f"Bert Tokens: {bert_tokens}\nTime Taken: {bert_time} seconds\n")
show_frequencies(bert_tokens, "Bert")
```

```
print(f"XLNet Tokens: {xlnet_tokens}\nTime Taken: {xlnet_time} seconds\n")
show_frequencies(xlnet_tokens, "XLNet")
```

---

5 Congratulations! You have completed the lab

5.1 Authors

Roodra Kanwar is completing his MS in CS specializing in big data from Simon Fraser University. He has previous experience working with machine learning and as a data engineer.

Joseph Santarcangelo has a Ph.D. in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Vicky Kuo is completing her Master's degree in IT at York University with scholarships. Her master's thesis explores the optimization of deep learning algorithms, employing an innovative approach to scrutinize and enhance neural network structures and performance.


```
{## Change Log}

{|Date (YYYY-MM-DD)|Version|Changed By|Change Description|}           {|-|-|-|-|}
{|2023-10-02|0.1|Roodra|Created Lab Template|}     {|2023-10-03|0.1|Vicky|Revised
the Lab|}
```