Inicialización

In [17]:
```
import logging

import numpy as np
import pandas as pd

import torch
import transformers

from tqdm.auto import tqdm
from collections.abc import Iterable
```

Cargar datos

Carga los datos de texto del archivo 'imdb_reviews_small.tsv'.

Se trata de un archivo de valores separados por tabuladores (TSV), lo cual significa que cada uno de los campos está separado por tabuladores (en lugar de por comas como has visto en otros ejercicios de Practicum).

In [24]:
```
data = pd.read_csv("/datasets/imdb_reviews_small.tsv", delimiter='\t')
```

Tokenizador BERT

Crear el tokenizador BERT a partir de un modelo previamente entrenado que se llama 'bert-base-uncased' en transformadores. Puedes echar un vistazo a su descripción general aquí. Puedes encontrar más detalles aquí.

In [10]:
```
tokenizer = transformers.BertTokenizer.from_pretrained('bert-base-uncased')
```

Hay un ejemplo de cómo obtener tokens para un solo texto.

Puedes usarlo para procesar todos los datos que cargaste anteriormente. Como ya hay muchos textos, y es probable que los proceses en un bucle, las longitudes mínimas/máximas de los vectores se pueden calcular de dos formas: dentro de un bucle o después de un bucle.

En el último caso, los vectores de identificadores numéricos de tokens (ids) y máscaras de atención (attention_mask) se deben almacenar en dos listas separadas. Se pueden llamar ids_list y atencion_mask_list, respectivamente. El primer caso te permite evitar la creación de esas listas, a menos que desees utilizarlas con otra finalidad, por ejemplo, para propagarlas en un modelo BERT. No se requiere en este ejercicio, pero se requerirá en el proyecto. Teniendo en cuenta lo anteriormente dicho, es posible que desees combinar ambas formas para calcular las longitudes mínimas/máximas de los vectores para tokens y máscaras de atención, así como conservar el

resultado del tokenizador para su posterior procesamiento. Solo considera que no tiene mucho sentido mantener vectores de más de 512 elementos, ya que esta es la longitud máxima de vectores que BERT puede aceptar.

In [11]:
```
# textos a tokens
text = 'It is very handy to use transformers' # (Es muy práctico utilizar transformador

# agregar este truco para suprimir las advertencias de salidas largas
# normalmente no es necesario, pero en este caso nos gustaría explorar# ¿cuál es la long
# por lo tanto, no truncamos la salida (ids) a la max_length
# con los parámetros max_length=max_length y truncation=True
logging.getLogger("transformers.tokenization_utils").setLevel(logging.ERROR)

ids = tokenizer.encode(text.lower(), add_special_tokens=True)

# padding (agregar ceros al vector para hacer que su longitud sea igual a n)
n = 512
padded = np.array(ids[:n] + [0]*(n - len(ids)))

# crear la máscara de atención para distinguir los tokens que nos interesan
attention_mask = np.where(padded != 0, 1, 0)
```

In [12]:
```
print(ids)
```

```
[101, 2009, 2003, 2200, 18801, 2000, 2224, 19081, 102]
```

In [13]:
```
print(padded)
```

```
[   101   2009   2003   2200 18801   2000   2224 19081    102      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0      0      0
```

```
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0]
```

In [14]:

```
print(attention_mask)
```

```
[1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Escribe tu código para tokenizar los datos de texto cargados.

In [27]:

```
def tokenize_with_bert(texts):
    # Verificar si texts es iterable
    if not isinstance(texts, Iterable):
        raise TypeError("El argumento 'texts' debe ser iterable.")

    ids_list = []
    attention_mask_list = []

    min_tokenized_text_length = float('inf')
    max_tokenized_text_length = 0

    for text in tqdm(texts, desc="Tokenizing"):
        # Tokenizar el texto
        encoded_dict = tokenizer.encode_plus(
                            text,                      # Texto a tokenizar
                            add_special_tokens = True, # Añadir tokens especiales para
                            max_length = 512,          # Máxima longitud de los tokens
                            padding = 'max_length',    # Rellenar los tokens hasta la l
                            truncation = True,         # Truncar ejemplos a max_length
                            return_attention_mask = True,  # Obtener la máscara de aten
                            return_tensors = 'pt',     # Devolver tensores de PyTorch
                       )

        # Obtener los IDs de los tokens y la máscara de atención
        ids = encoded_dict['input_ids']
        attention_mask = encoded_dict['attention_mask']

        # Actualizar la longitud mínima y máxima de los vectores
```

```
            min_tokenized_text_length = min(min_tokenized_text_length, ids.size(1))
            max_tokenized_text_length = max(max_tokenized_text_length, ids.size(1))

            # Agregar los IDs de los tokens y la máscara de atención a las listas
            ids_list.append(ids)
            attention_mask_list.append(attention_mask)

        print(f'La longitud mínima de los vectores: {min_tokenized_text_length}')
        print(f'La longitud máxima de los vectores: {max_tokenized_text_length}')

        return ids_list, attention_mask_list
```

Ejecuta el tokenizador para todos los datos. Puede llevar algún tiempo ya que

In [28]:

```
# Convertir la columna 'review' de data a una lista de reseñas
texts = data['review'].tolist()

# Llamar a la función tokenize_with_bert con la lista de reseñas
ids_list, attention_mask_list = tokenize_with_bert(texts=texts)
```

```
La longitud mínima de los vectores: 512
La longitud máxima de los vectores: 512
```

In [ ]: