Przetwarzanie języka naturalnego

Wstęp

Obecnie najpopularniejsze model służące do przetwarzania języka naturalnego wykorzystują architekturę transformacyjną. Istnieje kilka bibliotek, implementujących tę architekturę, ale w kontekście NLP najczęściej wykorzystuje się Huggingface transformers.

Biblioteka ta poza samym kodem źródłowym, zawiera szereg innych elementów. Do najważniejszych z nich należą:

- modele - olbrzymia i ciągle rosnąca liczba gotowych modeli, których możemy użyć do rozwiązywania wielu problemów z dziedziny NLP (ale również w zakresie rozpoznawania mowy, czy przetwarzania obrazu),
- zbiory danych - bardzo duży katalog przydatnych zbiorów danych, które możemy w prosty sposób wykorzystać do trenowania własnych modeli NLP (oraz innych modeli).

Weryfikacja dostępności GPU

Trening modeli NLP wymaga dostępu do akceleratorów sprzętowych, przyspieszających uczenie sieci neuronowych. Jeśli nasz komputer nie jest wyposażony w GPU, to możemy skorzystać ze środowiska Google Colab.

W tym środowisku możemy wybrać akcelerator spośród GPU i TPU.

Sprawdźmy, czy mamy dostęp do środowiska wyposażonego w akcelerator NVidii:

In [1]:
```
!nvidia-smi
```

```
Fri Dec 22 16:22:53 2023
+-----------------------------------------------------------------------------
-------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Vers
ion: 12.2     |
|-----------------------------------------+----------------------+--------
-------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatil
e Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Uti
l  Compute M. |
|                                         |                      |
MIG M. |
|=========================================+======================+========
==============|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |
0 |
| N/A   55C    P8              10W /  70W |      0MiB / 15360MiB |      0%
Default |
|                                         |                      |
N/A |
+-----------------------------------------+----------------------+--------
-------------+

+-----------------------------------------------------------------------------
-------------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name
GPU Memory |
|        ID   ID
Usage      |
|=============================================================================
==============|
|  No running processes found
|
+-----------------------------------------------------------------------------
-------------+
```

Jeśli akcelerator jest niedostępny (polecenie skończyło się błędem), to zmieniamy środowisko wykonawcze wybierając z menu "Środowisko wykonawcze" -> "Zmień typ środowiska wykonawczego" -> GPU.

Podpięcie dysku Google

Kolejnym elementem przygotowań, który jest opcjonalny, jest dołączenie własnego dysku Google Drive do środowiska Colab. Dzięki temu możliwe jest zapisywanie wytrenowanych modeli, w trakcie procesu treningu, na "zewnętrznym" dysku. Jeśli Google Colab doprowadzi do przerwania procesu treningu, to mimo wszystko pliki, które udało się zapisać w trakcie treningu nie przepadną. Możliwe będzie wznowienie treningu już na częściowo wytrenowanym modelu.

W tym celu montujemy dysk Google w Colabie. Wymaga to autoryzacji narzędzia Colab w Google Drive.

In [2]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

Po podmontowaniu dysku mamy dostęp do całej zawartości Google Drive. Wskazując miejsce zapisywania danych w trakcie treningu należy wskazać ścieżkę zaczynającą się od `/content/gdrive` , ale należy wskazać jakiś podkatalog w ramach naszej przestrzeni dyskowej. Pełna ścieżka może mieć postać `/content/gdrive /MyDrive/output` . Przed uruchomieniem treningu warto sprawdzić, czy dane zapisują się na dysku.

Instalacja bibliotek Pythona

Następnie zainstalujemy wszystkie niezbędne biblioteki. Poza samą biblioteką `transformers` , instalujemy również biblioteki do zarządzania zbiorami danych `datasets` , bibliotekę definiującą wiele metryk wykorzystywanych w algorytmach AI `evaluate` oraz dodatkowe narzędzia takie jak `sacremoses` oraz `sentencepiece` .

In [3]:
```
!pip install --no-cache-dir transformers==4.35.2 sacremoses==0.1.1 datase
```

```
Requirement already satisfied: transformers==4.35.2 in /usr/local/lib/pyth
on3.10/dist-packages (4.35.2)
Collecting sacremoses==0.1.1
  Downloading sacremoses-0.1.1-py3-none-any.whl (897 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 897.5/897.5 kB 5.7 MB/s eta
0:00:00
Collecting datasets==2.15.0
  Downloading datasets-2.15.0-py3-none-any.whl (521 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 521.2/521.2 kB 12.5 MB/s eta
0:00:00
Collecting evaluate==0.4.1
  Downloading evaluate-0.4.1-py3-none-any.whl (84 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 161.5 MB/s eta
0:00:00
Collecting sentencepiece==0.1.99
  Downloading sentencepiece-0.1.99-cp310-cp310-manylinux_2_17_x86_64.manyl
inux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 16.4 MB/s eta 0:0
0:00
Collecting accelerate==0.24.1
  Downloading accelerate-0.24.1-py3-none-any.whl (261 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 261.4/261.4 kB 20.4 MB/s eta
0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers==4.35.2) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/
lib/python3.10/dist-packages (from transformers==4.35.2) (0.19.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/di
st-packages (from transformers==4.35.2) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.1
0/dist-packages (from transformers==4.35.2) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/di
st-packages (from transformers==4.35.2) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python
3.10/dist-packages (from transformers==4.35.2) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from transformers==4.35.2) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/py
thon3.10/dist-packages (from transformers==4.35.2) (0.15.0)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python
3.10/dist-packages (from transformers==4.35.2) (0.4.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dis
t-packages (from transformers==4.35.2) (4.66.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-pac
kages (from sacremoses==0.1.1) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-pa
ckages (from sacremoses==0.1.1) (1.3.2)
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10
/dist-packages (from datasets==2.15.0) (10.0.1)
Collecting pyarrow-hotfix (from datasets==2.15.0)
  Downloading pyarrow_hotfix-0.6-py3-none-any.whl (7.9 kB)
Collecting dill<0.3.8,>=0.3.0 (from datasets==2.15.0)
  Downloading dill-0.3.7-py3-none-any.whl (115 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.3/115.3 kB 166.4 MB/s et
a 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-pa
ckages (from datasets==2.15.0) (1.5.3)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-pa
ckages (from datasets==2.15.0) (3.4.1)
Collecting multiprocess (from datasets==2.15.0)
```

```
  Downloading multiprocess-0.70.15-py310-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 190.9 MB/s et
a 0:00:00
Requirement already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in /usr/
local/lib/python3.10/dist-packages (from datasets==2.15.0) (2023.6.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-p
ackages (from datasets==2.15.0) (3.9.1)
Collecting responses<0.19 (from evaluate==0.4.1)
  Downloading responses-0.18.0-py3-none-any.whl (38 kB)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-pa
ckages (from accelerate==0.24.1) (5.9.5)
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.10/
dist-packages (from accelerate==0.24.1) (2.1.0+cu121)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/
dist-packages (from aiohttp->datasets==2.15.0) (23.1.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/pytho
n3.10/dist-packages (from aiohttp->datasets==2.15.0) (6.0.4)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10
/dist-packages (from aiohttp->datasets==2.15.0) (1.9.4)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python
3.10/dist-packages (from aiohttp->datasets==2.15.0) (1.4.1)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.
10/dist-packages (from aiohttp->datasets==2.15.0) (1.3.1)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/p
ython3.10/dist-packages (from aiohttp->datasets==2.15.0) (4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/li
b/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformer
s==4.35.2) (4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/
python3.10/dist-packages (from requests->transformers==4.35.2) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/d
ist-packages (from requests->transformers==4.35.2) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python
3.10/dist-packages (from requests->transformers==4.35.2) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python
3.10/dist-packages (from requests->transformers==4.35.2) (2023.11.17)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-pac
kages (from torch>=1.10.0->accelerate==0.24.1) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from torch>=1.10.0->accelerate==0.24.1) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-pa
ckages (from torch>=1.10.0->accelerate==0.24.1) (3.1.2)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/
dist-packages (from torch>=1.10.0->accelerate==0.24.1) (2.1.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/py
thon3.10/dist-packages (from pandas->datasets==2.15.0) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/d
ist-packages (from pandas->datasets==2.15.0) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.1->pandas->datasets==2.15.0) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.1
0/dist-packages (from jinja2->torch>=1.10.0->accelerate==0.24.1) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/d
ist-packages (from sympy->torch>=1.10.0->accelerate==0.24.1) (1.3.0)
Installing collected packages: sentencepiece, sacremoses, pyarrow-hotfix,
dill, responses, multiprocess, accelerate, datasets, evaluate
Successfully installed accelerate-0.24.1 datasets-2.15.0 dill-0.3.7 evalua
te-0.4.1 multiprocess-0.70.15 pyarrow-hotfix-0.6 responses-0.18.0 sacremos
es-0.1.1 sentencepiece-0.1.99
```

Mając zainstalowane niezbedne bilioteki, możemy skorzystać z wszystkich modeli i zbiorów danych zarejestrowanych w katalogu.

Typowym sposobem użycia dostępnych modeli jest:

- *wykorzystanie gotowego modelu*, który realizuje określone zadanie, np. analizę senetymentu w języku angielskim - model tego rodzaju nie musi być trenowywany, wystarczy go uruchomić aby uzyskać wynik klasyfikacji (można to zobaczyć w demo pod wskazanym linkiem),
- *wykorzystanie modelu bazowego*, który jest dotrenowywany do określonego zadania; przykładem takiego modelu jest HerBERT base, który uczony był jako maskowany model języka. Żeby wykorzystać go do konkretnego zadania, musimy wybrać dla niego "głowę klasyfikacyjną" oraz dotrenować na własnym zbiorze danych.

Modele tego rodzaju różnią się od siebie, można je załadować za pomocą wspólnego interfejsu, ale najlepiej jest wykorzystać jedną ze specjalizowanych klas, dostosowanych do zadania, które chcemy zrealizować. Zaczniemy od załadowania modelu BERT base - jednego z najbardziej popularnych modeli, dla języka angielskiego. Za jego pomocą będziemy odgadywać brakujące wyrazy w tekście. Wykorzystamy do tego wywołanie `AutoModelForMaskedLM` .

In [4]:
```
from transformers import AutoModelForMaskedLM, AutoTokenizer

model = AutoModelForMaskedLM.from_pretrained("bert-base-cased")
```

```
config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/436M [00:00<?, ?B/s]
Some weights of the model checkpoint at bert-base-cased were not used when initializing BertForMaskedLM: ['cls.seq_relationship.weight', 'bert.pooler.dense.bias', 'cls.seq_relationship.bias', 'bert.pooler.dense.weight']
- This IS expected if you are initializing BertForMaskedLM from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
```

Załadowany model jest modułem PyTorcha. Możemy zatem korzystać z API tej biblioteki. Możemy np. sprawdzić ile parametrów ma model BERT base:

In [5]:
```
count = sum(p.numel() for p in model.parameters() if p.requires_grad)

'{:,}'.format(count).replace(',', ' ')
```

Out[5]:
```
'108 340 804'
```

Widzimi zatem, że nasz model jest bardzo duży - zawiera ponad 100 milionów parametrów, a jest to tzw. model bazowy. Modele obecnie wykorzystywane mają jeszcze więcej parametrów - duże modele językowe, takie jak ChatGPT posiadają więcej niż 100 miliardów parametrów.

Możemy również podejrzeć samą strukturę modelu.

In [6]:
```
model
```

Out[6]:
```
BertForMaskedLM(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(28996, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=Tr
ue)
              (key): Linear(in_features=768, out_features=768, bias=Tru
e)
              (value): Linear(in_features=768, out_features=768, bias=Tr
ue)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=Tr
ue)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affi
ne=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=Tru
e)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=Tru
e)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine
=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (cls): BertOnlyMLMHead(
    (predictions): BertLMPredictionHead(
      (transform): BertPredictionHeadTransform(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (transform_act_fn): GELUActivation()
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=Tru
e)
      )
      (decoder): Linear(in_features=768, out_features=28996, bias=True)
    )
  )
)
```

Tokenizacja tekstu

Załadowanie samego modelu nie jest jednak wystarczające, żeby zacząć go wykorzystywać. Musimy mieć mechanizm zamiany tekstu (łańcucha znaków), na ciąg tokenów, należących do określonego słownika. W trakcie treningu modelu, słownik ten jest określany (wybierany w sposób algorytmiczny) przed właściwym treningiem sieci neuronowej. Choć możliwe jest jego późniejsze rozszerzenie (douczenie na danych treningowych, pozwala również uzyskać reprezentację brakujących tokenów), to zwykle wykorzystuje się słownik w postaci, która została określona przed treningiem sieci neuronowej. Dlatego tak istotne jest wskazanie właściwego słownika dla tokenizera dokonującego podziału tekstu.

Biblioteka posiada klasę `AutoTokenizer`, która akceptuje nazwę modelu, co pozwala automatycznie załadować słownik korespondujący z wybranym modelem sieci neuronowej. Trzeba jednak pamiętać, że jeśli używamy 2 modeli, to każdy z nich najpewniej będzie miał inny słownik, a co za tym idzie muszą one mieć własne instancje klasy `Tokenizer`.

In [7]:
```
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
tokenizer
```

```
tokenizer_config.json:   0%|          | 0.00/29.0 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/213k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/436k [00:00<?, ?B/s]
```

Out[7]:
```
BertTokenizerFast(name_or_path='bert-base-cased', vocab_size=28996, mode
l_max_length=512, is_fast=True, padding_side='right', truncation_side='r
ight', special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_
token': '[PAD]', 'cls_token': '[CLS]', 'mask_token': '[MASK]'}, clean_up
_tokenization_spaces=True),  added_tokens_decoder={
        0: AddedToken("[PAD]", rstrip=False, lstrip=False, single_word=F
alse, normalized=False, special=True),
        100: AddedToken("[UNK]", rstrip=False, lstrip=False, single_word
=False, normalized=False, special=True),
        101: AddedToken("[CLS]", rstrip=False, lstrip=False, single_word
=False, normalized=False, special=True),
        102: AddedToken("[SEP]", rstrip=False, lstrip=False, single_word
=False, normalized=False, special=True),
        103: AddedToken("[MASK]", rstrip=False, lstrip=False, single_wor
d=False, normalized=False, special=True),
}
```

Tokenizer posługuje się słownikiem o stałym rozmiarze. Podowuje to oczywiście, że nie wszystkie wyrazy występujące w tekście, będą się w nim znajdowały. Co więcej, jeśli użyjemy tokenizera do podziału tekstu w innym języku, niż ten dla którego został on stworzony, to taki tekst będzie dzielony na większą liczbę tokenów.

In [8]:
```
sentence1 = tokenizer.encode(
    "The quick brown fox jumps over the lazy dog.", return_tensors="pt"
)
print(sentence1)
print(sentence1.shape)
```

```
sentence2 = tokenizer.encode("Zażółć gęślą jaźń.", return_tensors="pt")
print(sentence2)
print(sentence2.shape)
```

```
tensor([[  101,  1109,  3613,  3058, 17594, 15457,  1166,  1103, 16688,  3
676,
           119,   102]])
torch.Size([1, 12])
tensor([[  101,   163,  1161, 28259,  7774, 20671,  7128,   176, 28221, 28
244,
         1233, 28213,   179,  1161, 28257, 19339,   119,   102]])
torch.Size([1, 18])
```

Korzystając z tokenizera dla języka angielsiego do podziału polskiego zdania, widzimy, że otrzymujemy znacznie większą liczbę tokenów. Żeby zobaczyć, w jaki sposób tokenizer dokonał podziału tekstu, możemy wykorzystać wywołanie `covert_ids_to_tokens` :

In [9]:
```
print("|".join(tokenizer.convert_ids_to_tokens(list(sentence1[0]))))
print("|".join(tokenizer.convert_ids_to_tokens(list(sentence2[0]))))
```

```
[CLS]|The|quick|brown|fox|jumps|over|the|lazy|dog|.|[SEP]
[CLS]|Z|##a|##ż|##ó|##ł|##ć|g|##ę|##ś|##l|##ą|j|##a|##ź|##ń|.|[SEP]
```

Widzimy, że dla jęzka angielskiego wszystkie wyrazy w zdaniu zostały przekształcone w pojedyncze tokeny. W przypadku zdania w języku polskim, zawierającego szereg znaków diakrytycznych sytuacja jest zupełnie inna - każdy znak został wyodrębniony do osobnego sub-tokenu. To, że mamy do czynienia z sub-tokenami sygnalizowane jest przez dwa krzyżyki poprzedzające dany sub-token. Oznaczają one, że ten sub-token musi być sklejony z porzedzającym go tokenem, aby uzyskać właściwy łańcuch znaków.

Zadanie 1 (0.5 punkt)

Wykorzystaj tokenizer dla modelu `allegro/herbert-base-cased` , aby dokonać tokenizacji tych samych zdań. Jakie wnioski można wyciągnąć przyglądając się sposobowi tokenizacji za pomocą różnych słowników?

In [10]:
```
tokenizer_herbert = AutoTokenizer.from_pretrained("allegro/herbert-base-c

sentence1 = tokenizer_herbert.encode(
    "The quick brown fox jumps over the lazy dog.", return_tensors="pt"
)
print(sentence1)
print(sentence1.shape)
print("|".join(tokenizer_herbert.convert_ids_to_tokens(list(sentence1[0])

sentence2 = tokenizer_herbert.encode("Zażółć gęślą jaźń.", return_tensors
print(sentence2)
print(sentence2.shape)
print("|".join(tokenizer_herbert.convert_ids_to_tokens(list(sentence2[0])
```

```
tokenizer_config.json:   0%|          | 0.00/229 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/472 [00:00<?, ?B/s]
vocab.json:   0%|          | 0.00/907k [00:00<?, ?B/s]
```

```
merges.txt:   0%|          | 0.00/556k [00:00<?, ?B/s]
special_tokens_map.json:   0%|          | 0.00/129 [00:00<?, ?B/s]
tensor([[    0,  7117, 22991,  4879, 25015,  1016,  3435,  1055,  2202,  4
952,
         1010,    83, 10259,  6854,  2050,  3852,  2065,  1031,  1899,
2]])
torch.Size([1, 20])
<s>|The</w>|qui|ck</w>|brow|n</w>|fo|x</w>|ju|mp|s</w>|o|ver</w>|the</w>|l
a|zy</w>|do|g</w>|.</w>|</s>
tensor([[    0,  2237,  7227,  1048,  7029, 46389,  2059,   272,  1059,  1
899,
             2]])
torch.Size([1, 11])
<s>|Za|żół|ć</w>|gę|ślą</w>|ja|ź|ń</w>|.</w>|</s>
```

Komentarz

Wykorzystując różne słowniki, możemy osiągnąć inne wyniki tokenizacji tekstu. Na przykład dla słownika w języku angielskim otrzymaliśmy odmienne rezultaty niż dla języka polskiego, HerBERT poradził sobie lepiej, oczywiście, w wypadku zdania po polsku - nie podzielił każdego polskiego znaku na osobny subtoken.

W wynikach tokenizacji poza wyrazami/tokenami występującymi w oryginalnym tekście pojawiają się jeszcze dodatkowe znaczniki `[CLS]` oraz `[SEP]` (albo inne znaczniki - w zależności od użytego słownika). Mają one specjalne znaczenie i mogą być wykorzystywane do realizacji specyficznych funkcji związanych z analizą tekstu. Np. reprezentacja tokenu `[CLS]` wykorzystywana jest w zadaniach klasyfikacji zdań. Z kolei token `[SEP]` wykorzystywany jest do odróżnienia zdań, w zadaniach wymagających na wejściu dwóch zdań (np. określenia, na ile zdania te są podobne do siebie).

Modelowanie języka

Modele pretrenowane w reżimie self-supervised learning (SSL) nie posiadają specjalnych zdolności w zakresie rozwiązywania konkretnych zadań z zakresu przetwarzania języka naturalnego, takich jak odpowiadanie na pytania, czy klasyfikacja tekstu (z wyjątkiem bardzo dużych modeli, takich jak np. GPT-3, których model językowy zdolny jest do predykcji np. sensownych odpowiedzi na pytania). Można je jednak wykorzystać do określania prawdopodobieństwa wyrazów w tekście, a tym samym do sprawdzenia, jaką wiedzę posiada określony model w zakresie znajomości języka, czy też ogólną wiedzę o świecie.

Aby sprawdzić jak model radzi sobie w tych zadaniach, możemy dokonać inferencji na danych wejściowych, w których niektóre wyrazy zostaną zastąpione specjalnymi symbolami maskującymi, wykorzystywanymi w trakcie pre-treningu modelu.

Należy mieć na uwadze, że różne modele mogą korzystać z różnych specjalnych sekwencji w trakcie pretreningu. Np. Bert korzysta z sekwencji `[MASK]`. Wygląd tokenu maskującego lub jego identyfikator możemy sprawdzić w pliku konfiguracji tokenizera dystrubowanym razem z modelem, albo odczytać wprost z instancji

tokenizera.

W pierwszej kolejności, spróbujemy uzupełnić brakujący wyraz w angielskim zdaniu.

In [11]:
```
sentence_en = tokenizer.encode(
    "The quick brown [MASK] jumps over the lazy dog.", return_tensors="pt
)
print("|".join(tokenizer.convert_ids_to_tokens(list(sentence_en[0]))))
target = model(sentence_en)
print(target.logits[0][4])
```

```
[CLS]|The|quick|brown|[MASK]|jumps|over|the|lazy|dog|.|[SEP]
tensor([-5.3489, -5.6063, -5.1303,  ..., -5.9625, -4.1559, -4.5403],
       grad_fn=<SelectBackward0>)
```

Ponieważ zdanie po stokenizowaniu uzupełniane jest znacznikiem `[CLS]`, to zamaskowane słowo znajduje się na 4 pozycji. Wywołanie `target.logits[0][4]` pokazuje tensor z rozkładem prawdopodobieństwa poszczególnych wyrazów, które zostało określone na podstawie parametrów modelu. Możemy wybrać wyrazy, które posiadają największe prawdopodobieństwo, korzystając z wywołania `torch.topk`:

In [12]:
```
import torch

top = torch.topk(target.logits[0][4], 5)
top
```

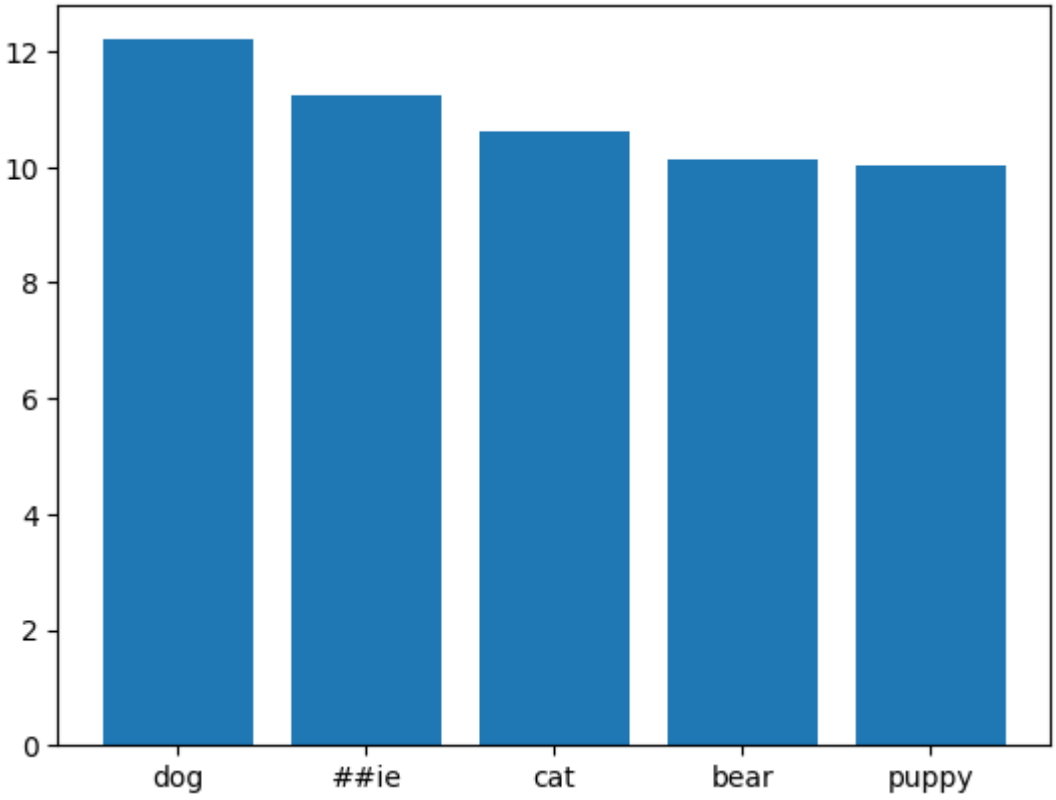
Out[12]:
```
torch.return_types.topk(
values=tensor([12.1982, 11.2289, 10.6009, 10.1278, 10.0120], grad_fn=<To
pkBackward0>),
indices=tensor([ 3676,  1663,  5855,  4965, 21566]))
```

Otrzymaliśmy dwa wektory - `values` zawierający składowe wektora wyjściowego sieci neuronowej (nieznormalizowane) oraz `indices` zawierający indeksy tych składowych. Na tej podstawie możemy wyświetlić wyraz, które według modelu są najbardziej prawdopodobnymi uzupełnieniami zamaskowanego wyrazu:

In [13]:
```
words = tokenizer.convert_ids_to_tokens(top.indices)
```

In [14]:
```
import matplotlib.pyplot as plt

plt.bar(words, top.values.detach().numpy())
```

Out[14]:
```
<BarContainer object of 5 artists>
```



Według modelu najbardziej prawdopodobnym uzupełnieniem brakującego wyrazu jest `dog` (a nie `fox`). Nieco zaskakujący może być drugi wyraz `##ie`, ale po dodaniu go do istniejącego tekstu otrzymamy zdanie: "The quick brownie jumps over the lazy dog", które również wydaje się sensowne (choć nieco zaskakujące).

Zadanie 2 (1.5 punkty)

Wykorzystując model `allegro/herbert-base-cased` zaproponuj zdania z jednym brakującym wyrazem, weryfikujące zdolność tego modelu do:

- odmiany przez polskie przypadki,
- uwzględniania długodystansowych związków w tekście,
- reprezentowania wiedzy o świecie.

Dla każdego problemu wymyśl po 3 zdania sprawdzające i wyświetl predykcję dla 5 najbardziej prawdopodobnych wyrazów.

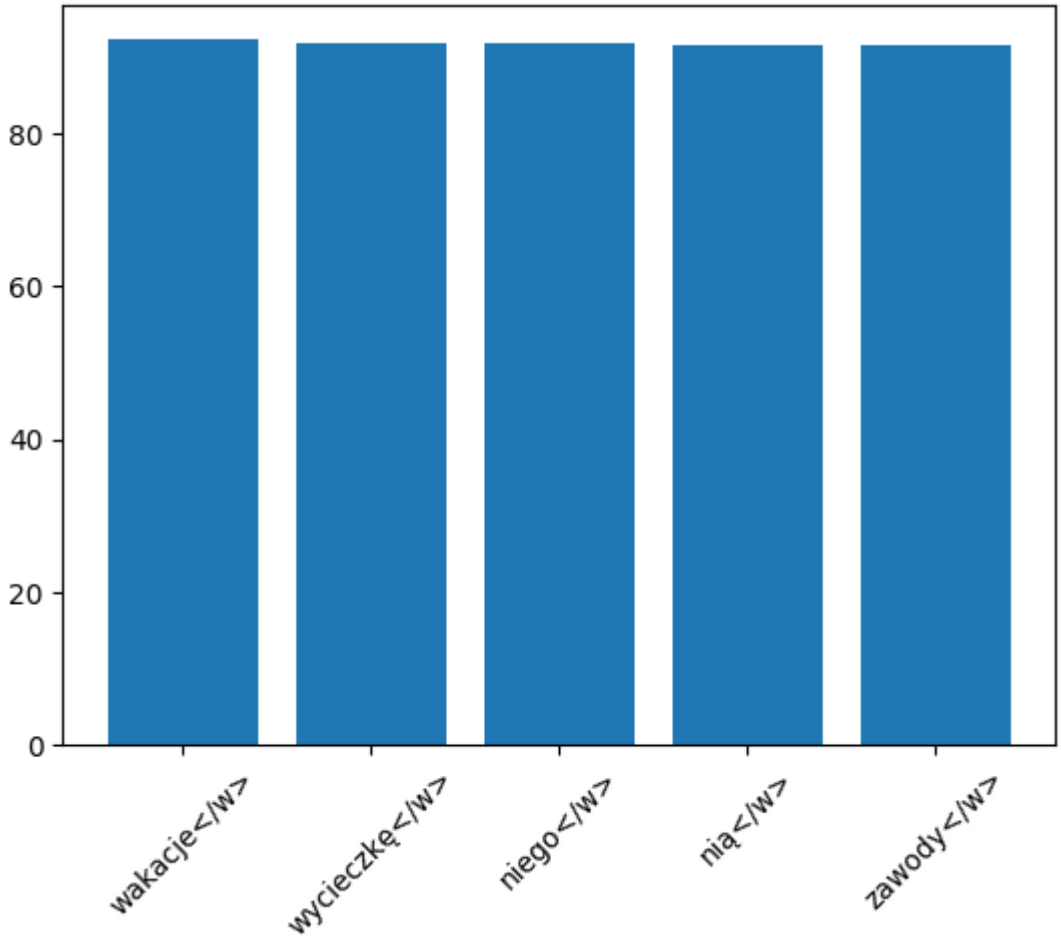
Możesz wykorzystać kod z funkcji `plot_words`, który ułatwi Ci wyświetlanie wyników. Zweryfikuj również jaki token maskujący wykorzystywany jest w tym modelu. Pamiętaj również o załadowaniu modelu `allegro/herbert-base-cased`.

Oceń zdolności modelu w zakresie wskazanych zadań.

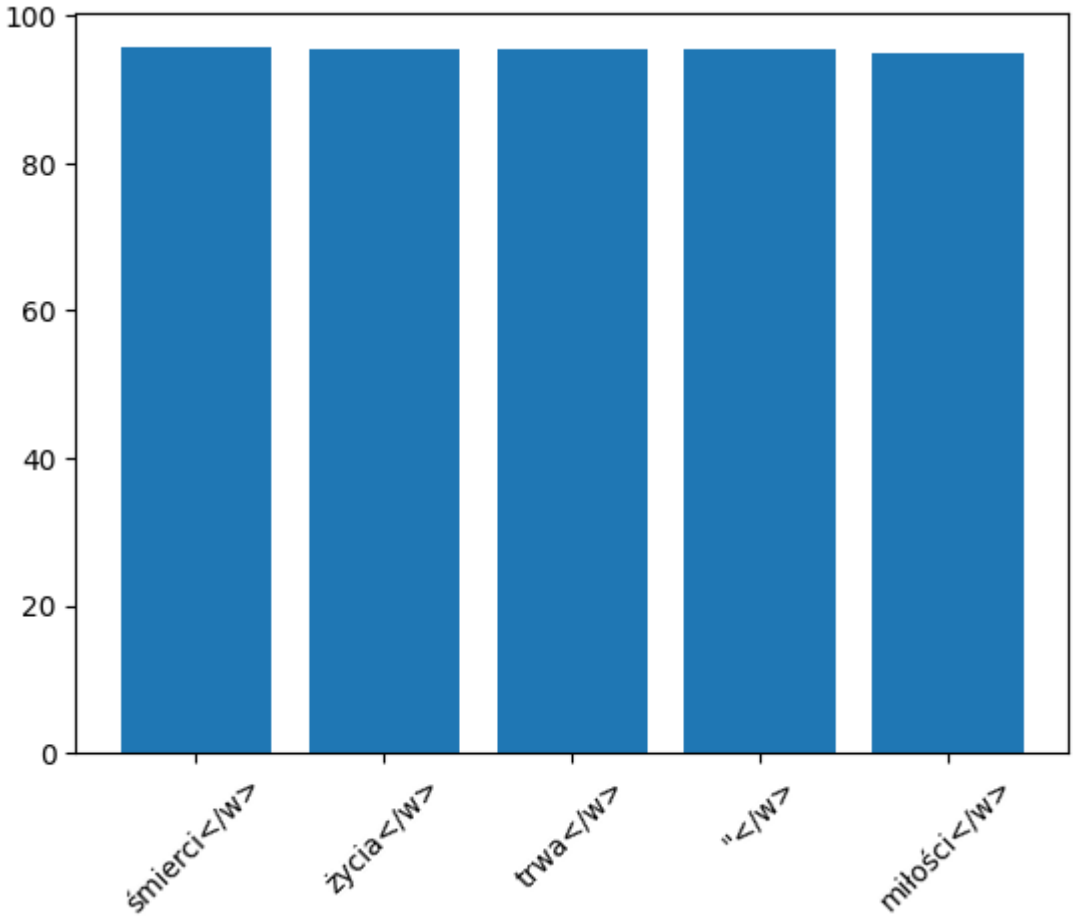
In [15]:
```
def plot_words(sentence, word_model, word_tokenizer, mask="[MASK]"):
    sentence_str = sentence
    sentence = word_tokenizer.encode(sentence, return_tensors="pt")
    tokens = word_tokenizer.convert_ids_to_tokens(list(sentence[0]))
    # print("|".join(tokens))
    target = word_model(sentence)
```

```
    top = torch.topk(target.logits[0][tokens.index(mask)], 5)
    words = word_tokenizer.convert_ids_to_tokens(top.indices)

    print(sentence_str)
    print(sentence_str.replace(mask, words[0]))

    plt.xticks(rotation=45)
    plt.bar(words, top.values.detach().numpy())
    plt.show()


MASK_TOKEN = "<mask>"
model_herbert = AutoModelForMaskedLM.from_pretrained("allegro/herbert-bas
sentences = {
    "conjugation": [
        f"Pojechałem na {MASK_TOKEN} w zeszłym tygodniu.",
        f"Piekło {MASK_TOKEN}.",
        f"Nie jestem twoją {MASK_TOKEN}.",
    ],
    "context": [
        f"Nigdzie się tak dobrze nie bawiłem, jak w {MASK_TOKEN}, zdobywa
        f"{MASK_TOKEN} to działanie odwrotne mnożenia, jedno z czterech p
            f"arytmetycznych, oznaczany zwykle za pomocą symbolu \"/\".",
        f"Wypełnij swoje {MASK_TOKEN}, zrób to co mówi przepowiednia.",
    ],
    "knowledge": [
        f"Największym państwem pod względem powierzchni jest {MASK_TOKEN}
        f"Pierwsze prawo Ohma mówi o tym, że napięcie jest proporcjonalne
        f"Woda składa się to związek chemiczny składający się z: wodoru i
    ]
}

for sentence_category in sentences:
    for sentence in sentences[sentence_category]:
        plot_words(sentence, model_herbert, tokenizer_herbert, MASK_TOKEN
        print("\n\n\n")
```

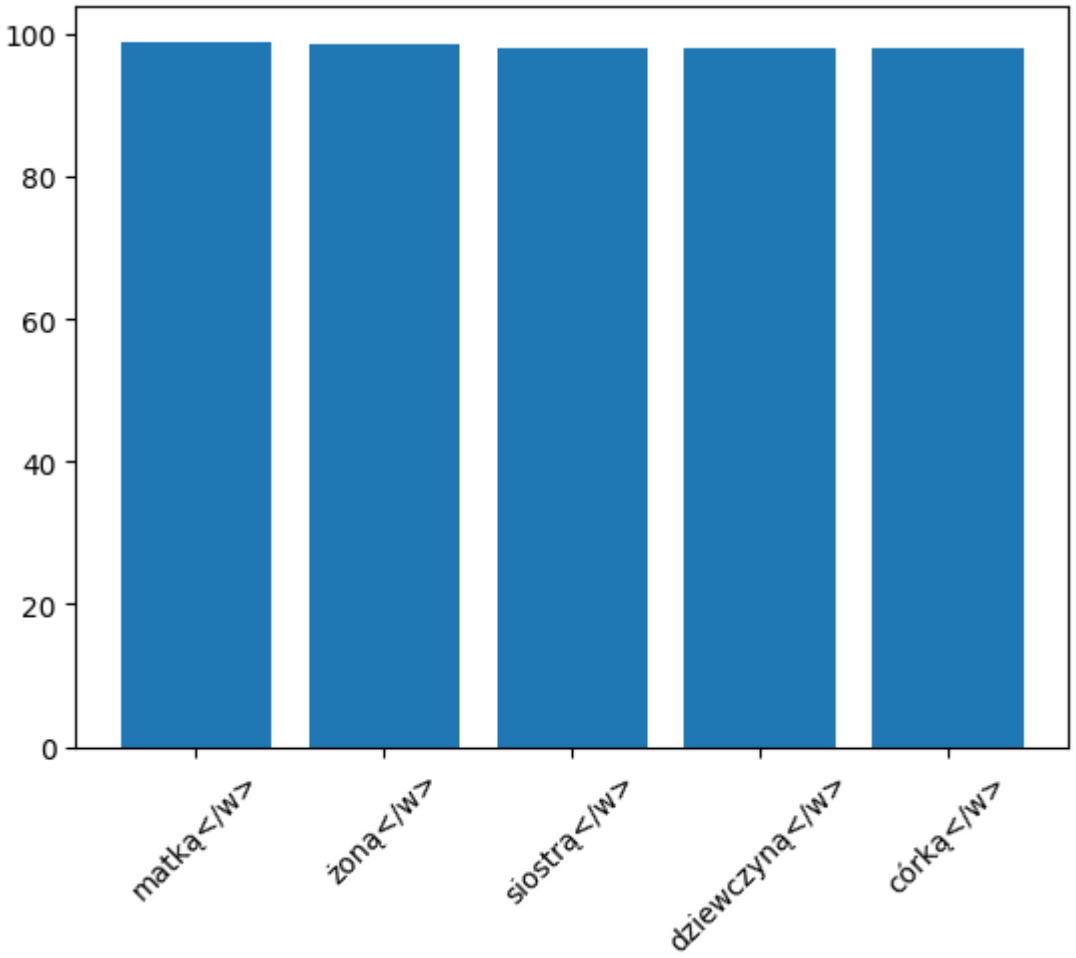
```
pytorch_model.bin:   0%|          | 0.00/654M [00:00<?, ?B/s]
Pojechałem na <mask> w zeszłym tygodniu.
Pojechałem na wakacje</w> w zeszłym tygodniu.
```



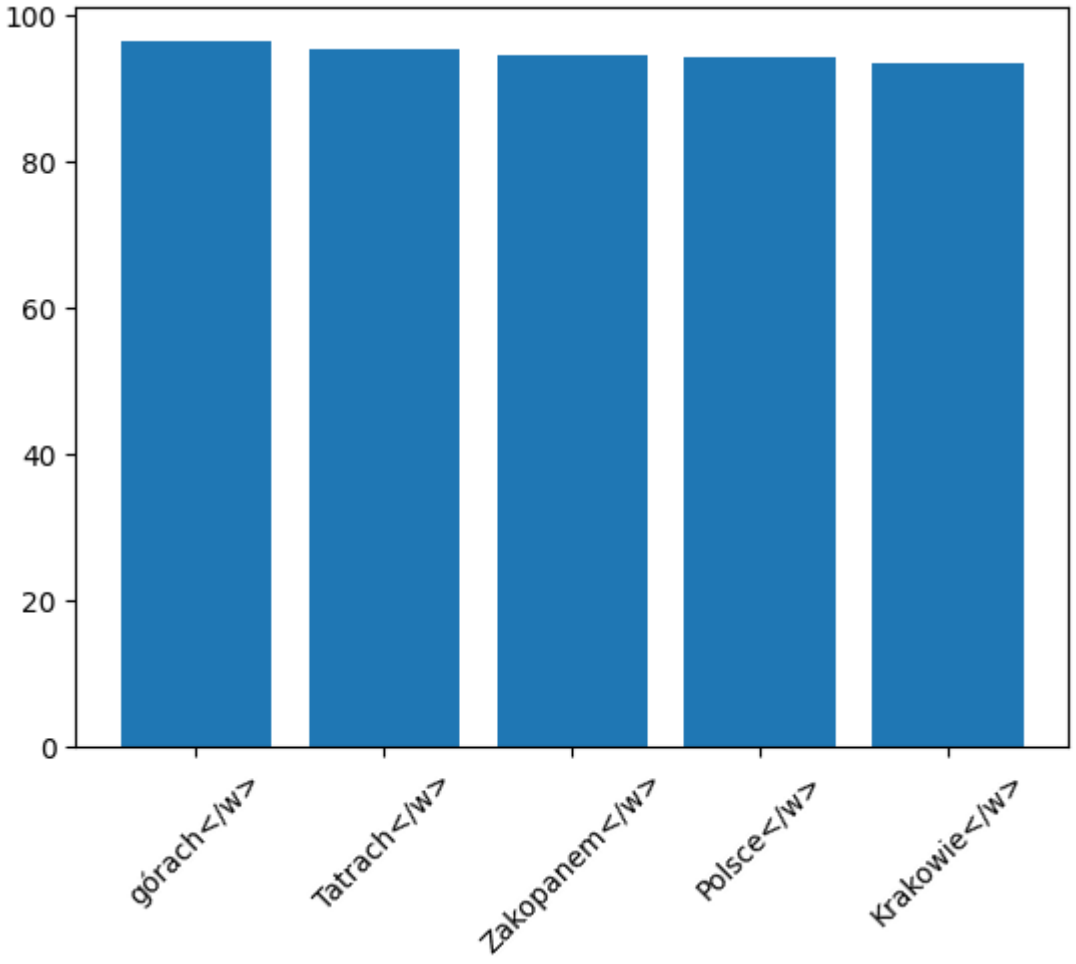
Piekło <mask>.
Piekło śmierci</w>.



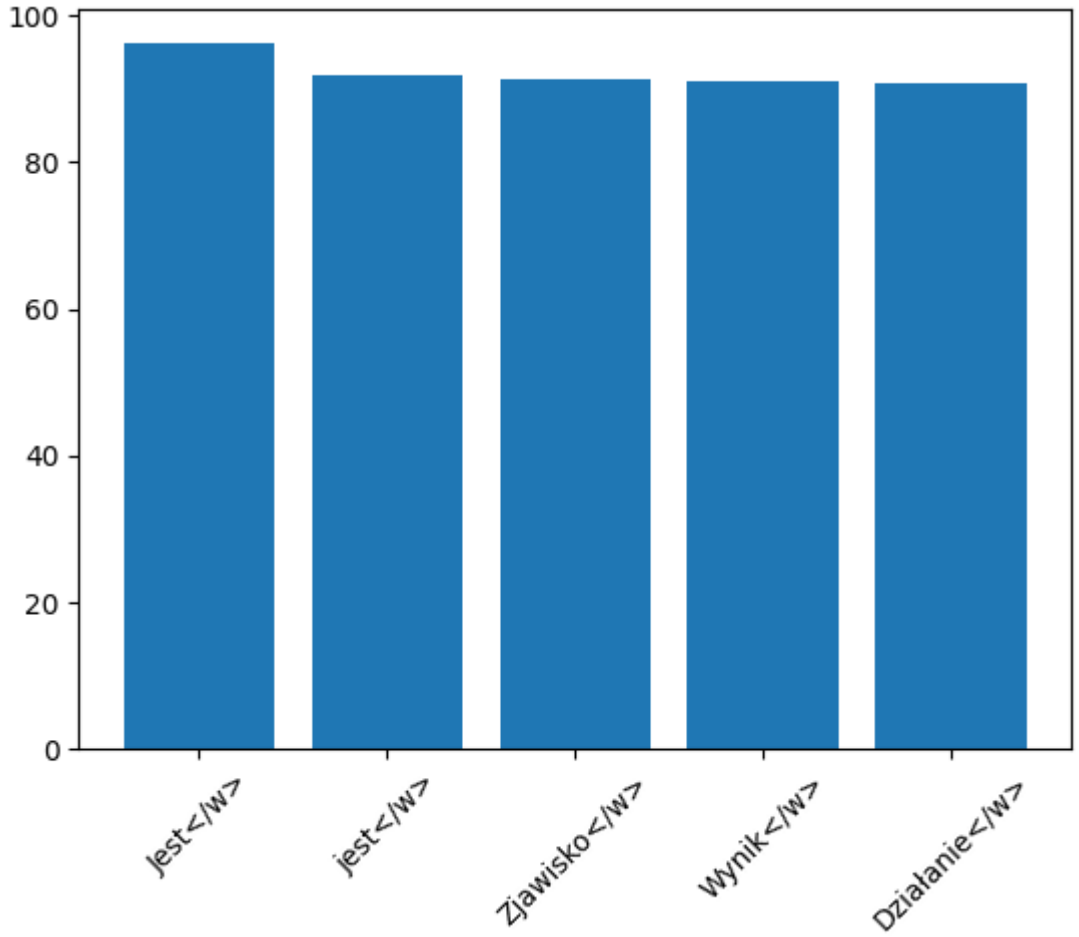
Nie jestem twoją <mask>.
Nie jestem twoją matką</w>.



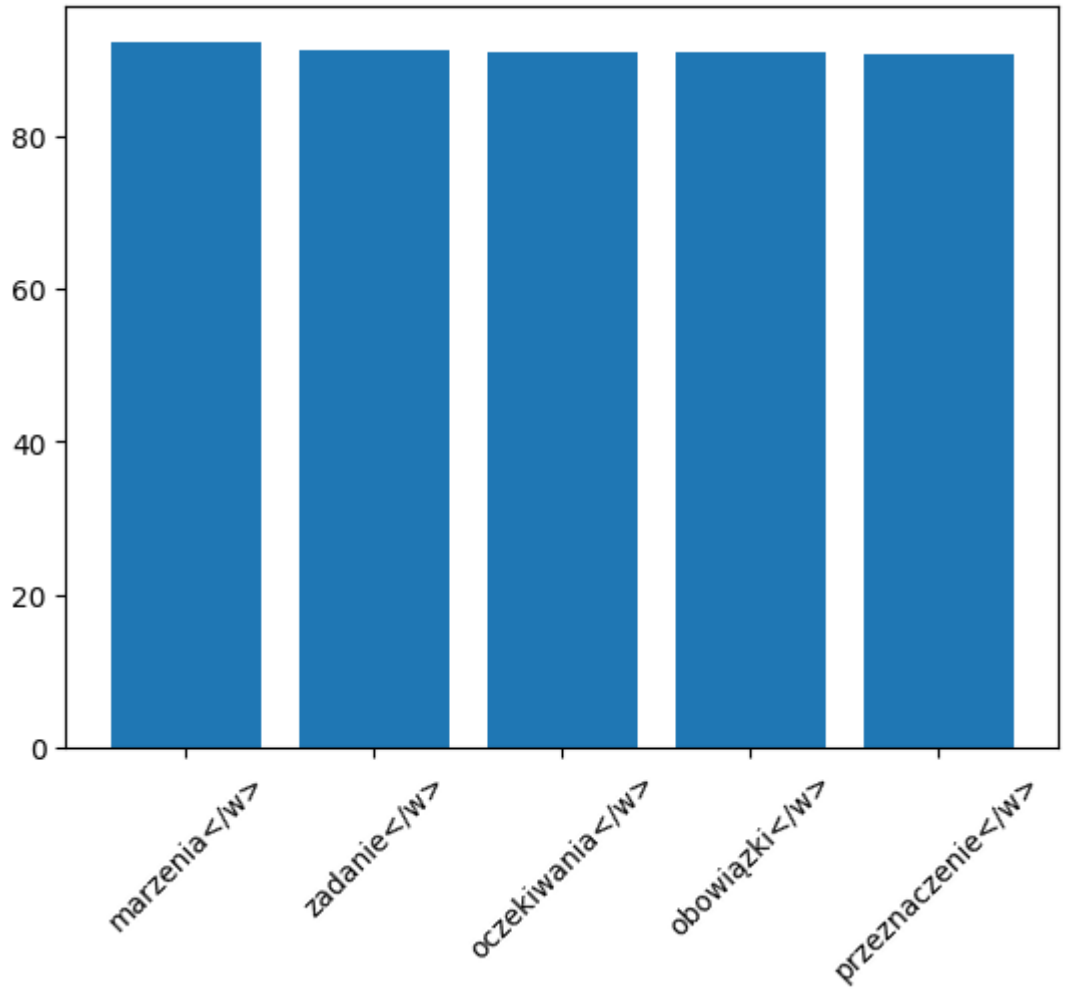
Nigdzie się tak dobrze nie bawiłem, jak w <mask>, zdobywając szczyt.
Nigdzie się tak dobrze nie bawiłem, jak w górach</w>, zdobywając szczyt.



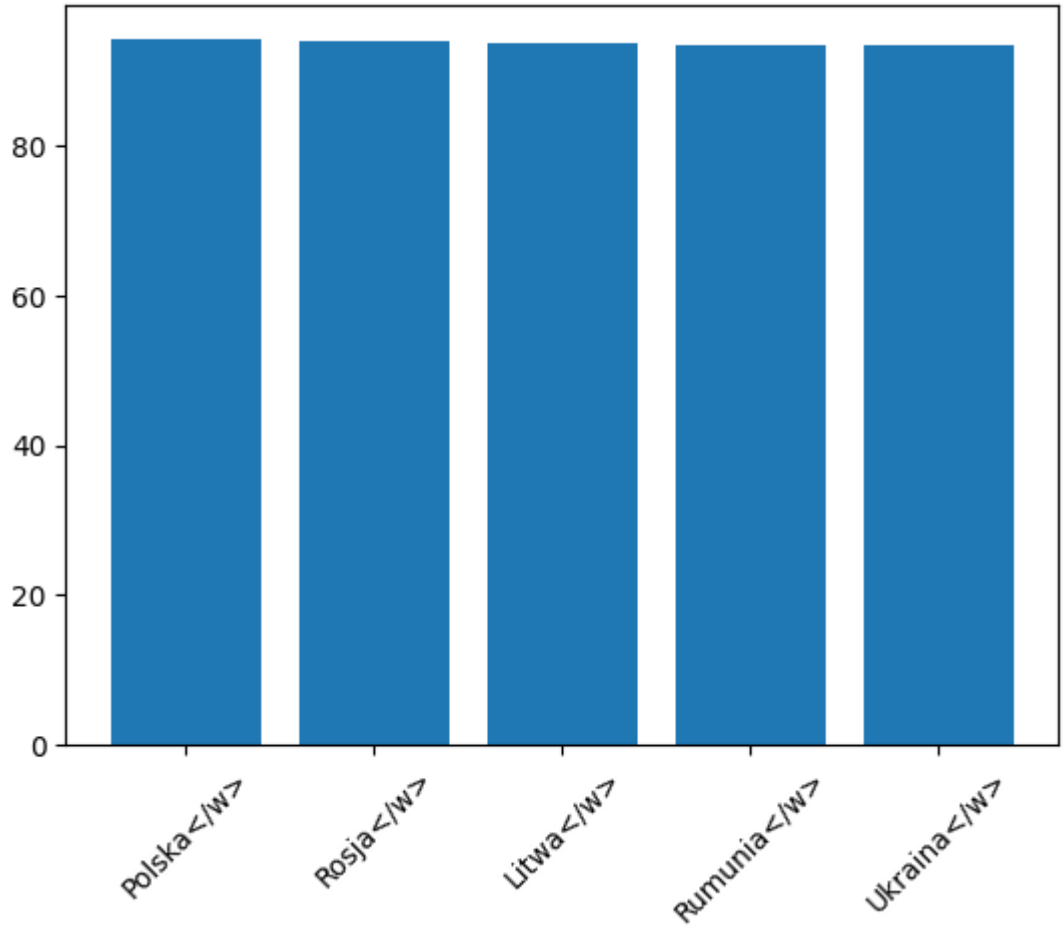
<mask> to działanie odwrotne mnożenia, jedno z czterech podstawowych opera
cjiarytmetycznych, oznaczany zwykle za pomocą symbolu "/".
Jest</w> to działanie odwrotne mnożenia, jedno z czterech podstawowych ope
racjiarytmetycznych, oznaczany zwykle za pomocą symbolu "/".



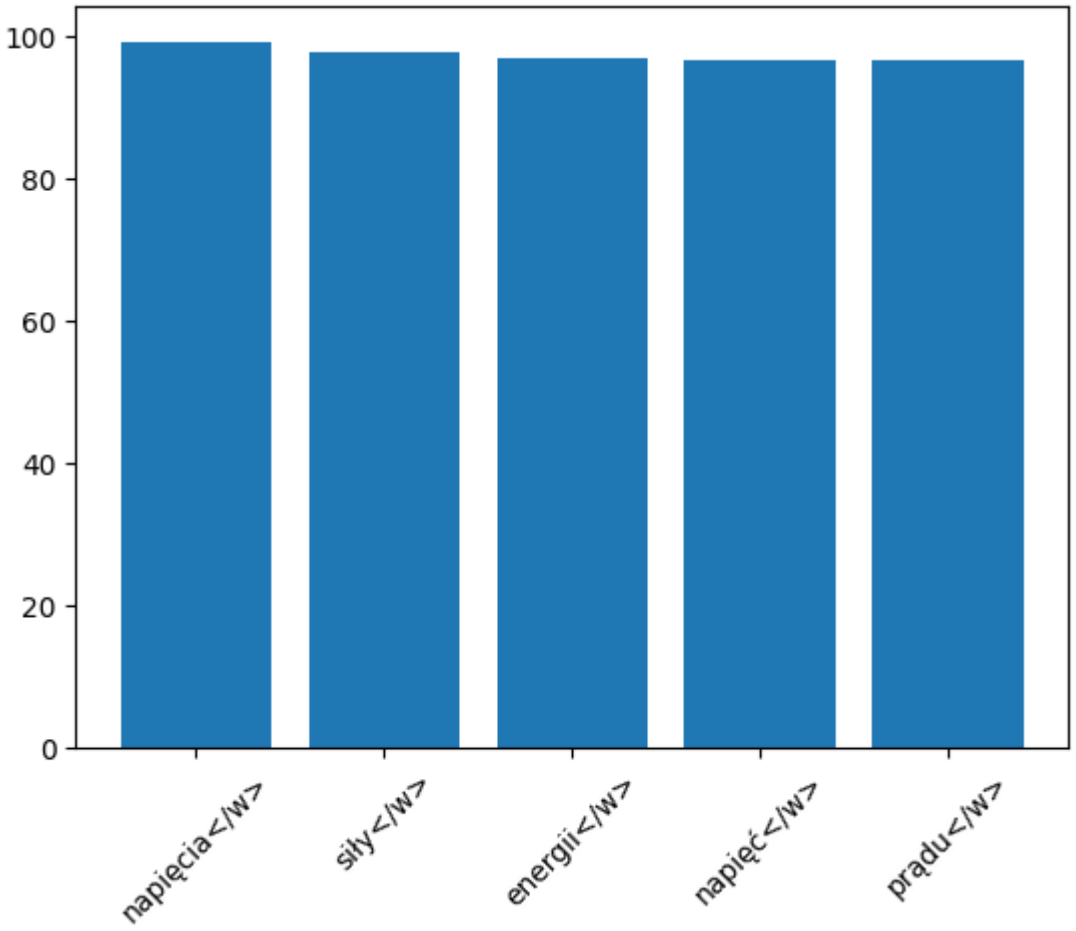
Wypełnij swoje <mask>, zrób to co mówi przepowiednia.
Wypełnij swoje marzenia</w>, zrób to co mówi przepowiednia.



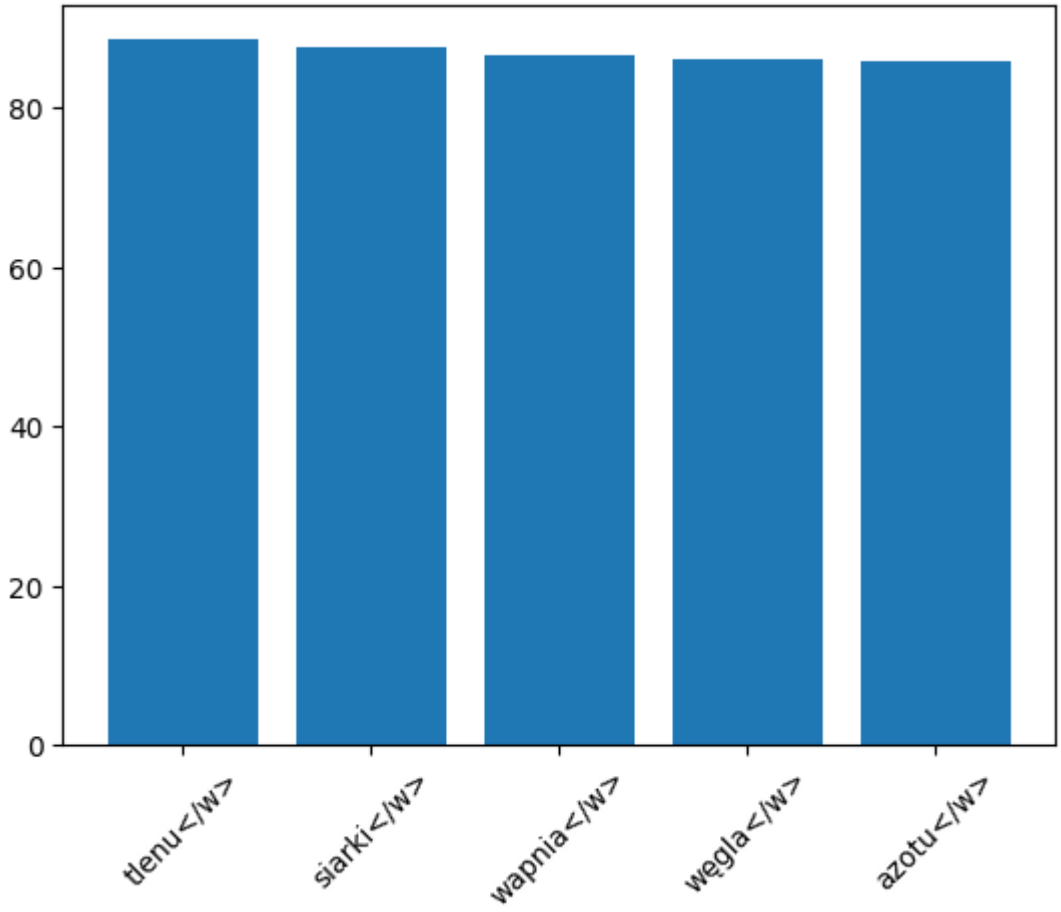

```
Największym państwem pod względem powierzchni jest <mask>.
Największym państwem pod względem powierzchni jest Polska</w>.
```



Pierwsze prawo Ohma mówi o tym, że napięcie jest proporcjonalne do iloczyn
u <mask> i rezystancji.
Pierwsze prawo Ohma mówi o tym, że napięcie jest proporcjonalne do iloczyn
u napięcia</w> i rezystancji.



Woda składa się to związek chemiczny składający się z: wodoru i <mask>.
Woda składa się to związek chemiczny składający się z: wodoru i tlenu</w>.



Komentarz

Model HerBERT bardzo dobrze sobie radzi z odmianą przez przypadki; obliczanie prawdopodobieństwa z konktekstu jest słabe; a jego wiedza ogólna jest także bardzo dobra, widać jego pewną wadę - model jest ewidentnie trenowany na języku polskim, stąd jego podświadome zamiłowanie do Polski :D

Klasyfikacja tekstu

Pierwszym zadaniem, które zrealizujemy korzystając z modelu HerBERT będzie klasyfikacja tekstu. Będzie to jednak dość nietypowe zadanie. O ile oczekiwanym wynikiem jest klasyfikacja binarna, czyli dość popularny typ klasyfikacji, o tyle dane wejściowe są nietypowe, gdyż są to pary: `(pytanie, kontekst)`. Celem algorytmu jest określenie, czy na zadane pytanie można odpowiedzieć na podstawie informacji znajdujących się w kontekście.

Model tego rodzaju jest nietypowy, ponieważ jest to zadanie z zakresu klasyfikacji par tekstów, ale my potraktujemy je jak zadanie klasyfikacji jednego tekstu, oznaczając jedynie fragmenty tekstu jako `Pytanie:` oraz `Kontekst:`. Wykorzystamy tutaj zdolność modeli transformacyjnych do automatycznego nauczenia się tego rodzaju znaczników, przez co proces przygotowania danych

będzie bardzo uproszczony.

Zbiorem danych, który wykorzystamy do treningu i ewaluacji modelu będzie PoQUAD - zbiór inspirowany angielskim SQuADem, czyli zbiorem zawierającym ponad 100 tys. pytań i odpowiadających im odpowiedzi. Zbiór ten powstał niedawno i jest jeszcze rozbudowywany. Zawiera on pytania, odpowiedzi oraz konteksty, na podstawie których można udzielić odpowiedzi.

W dalszej części laboratorium skoncentrujemy się na problemie odpowiadania na pytania.

Przygotowanie danych do klasyfikacji

Przygotowanie danych rozpoczniemy od sklonowania repozytorium zawierającego pytania i odpowiedzi.

In [16]:
```
from datasets import load_dataset

dataset = load_dataset("clarin-pl/poquad")
```

```
Downloading builder script:   0%|          | 0.00/5.35k [00:00<?, ?B/s]
Downloading readme:   0%|          | 0.00/317 [00:00<?, ?B/s]
Downloading data files:   0%|          | 0/2 [00:00<?, ?it/s]
Downloading data:   0%|          | 0.00/47.2M [00:00<?, ?B/s]
Downloading data:   0%|          | 0.00/6.29M [00:00<?, ?B/s]
Extracting data files:   0%|          | 0/2 [00:00<?, ?it/s]
Generating train split: 0 examples [00:00, ? examples/s]
Generating validation split: 0 examples [00:00, ? examples/s]
```

Sprawdźmy co znajduje się w zbiorze danych.

In [17]:
```
dataset
```

Out[17]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 46187
    })
    validation: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 5764
    })
})
```

Zbiór danych jest podzielony na dwie części: treningową i walidacyjną. Rozmiar części treningowej to ponad 46 tysięcy pytań i odpowiedzi, natomiast części walidacyjnej to ponad 5 tysięcy pytań i odpowiedzi.

Dane zbioru przechowywane są w plikach `poquad_train.json` oraz `poquad_dev.json` . Dostarczenie podziału na te grupy danych jest bardzo częstą praktyką w przypadku publicznych, dużych zbiorów danych, gdyż umożliwia porównywanie różnych modeli, korzystając z dokładnie takiego samego zestawu danych. Prawdopodobnie istnieje również zbiór `poquad_test.json` , który jednak

nie jest udostępniany publicznie. Tak jest w przypadku SQuADu - twórcy zbioru automatycznie ewaluują dostarczane modele, ale nie udstoępniaja zbioru testowego. Dzięki temu trudniej jest nadmiernie dopasować model do danych testowych.

Struktura każdej z dostępnych części jest taka sama. Zgodnie z powyższą informacją zawiera ona następujące elementy:

- `id` - identyfikator pary: pytanie - odpowiedź,
- `title` - tytuł artykułu z Wikipedii, na podstawie którego utworzono parę,
- `context` - fragment treści artykułu z Wikipedii, zawierający odpowiedź na pytanie,
- `question` - pytanie,
- `answers` - odpowiedzi.

Możemy wyświetlić kilka początkotwych wpisów części treningowej:

In [18]:
```
dataset['train']['question'][:5]
```

Out[18]:
```
['Co było powodem powrócenia konceptu porozumieniu monachijskiego?',
 'Pomiędzy jakimi stronami odbyło się zgromadzenie w sierpniu 1942 rok
u?',
 'O co ubiegali się polscy przedstawiciele podczas spotkania z sierpnia
1942 roku?',
 "Który z dyplomatów sprzeciwił się konceptowi konfederacji w listopadzi
e '42?",
 'Kiedy oficjalnie doszło do zawarcia porozumienia?']
```

In [19]:
```
dataset['train']['answers'][:5]
```

Out[19]:
```
[{'text': ['wymianą listów Ripka — Stroński'], 'answer_start': [117]},
 {'text': ['E. Beneša i J. Masaryka z jednej a Wł. Sikorskiego i E. Racz
yńskiego'],
  'answer_start': [197]},
 {'text': ['podpisanie układu konfederacyjnego'], 'answer_start': [31
5]},
 {'text': ['E. Beneš'], 'answer_start': [558]},
 {'text': ['20 listopada 1942'], 'answer_start': [691]}]
```

Niestety, autorzy zbioru danych, pomimo tego, że dane te znajdują się w źródłowym zbiorze danych, nie udostępniają dwóch ważnych informacji: o tym, czy można odpowiedzieć na dane pytanie oraz jak brzmi generatywna odpowiedź na pytanie. Dlatego póki nie zostanie to naprawione, będziemy dalej pracować z oryginalnymi plikami zbioru danych, które dostępne są na stronie opisującej zbiór danych: https://huggingface.co/datasets/clarin-pl/poquad/tree/main

Pobierz manualnie zbiory `poquad-dev.json` oraz `poquad-train.json` .

In [20]:
```
!wget https://huggingface.co/datasets/clarin-pl/poquad/raw/main/poquad-de
!wget https://huggingface.co/datasets/clarin-pl/poquad/resolve/main/poqua
```

```
--2023-12-22 16:24:50--  https://huggingface.co/datasets/clarin-pl/poquad/raw/main/poquad-dev.json
Resolving huggingface.co (huggingface.co)... 18.244.202.60, 18.244.202.73, 18.244.202.68, ...
Connecting to huggingface.co (huggingface.co)|18.244.202.60|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6286317 (6.0M) [text/plain]
Saving to: ‘poquad-dev.json’

poquad-dev.json     100%[===================>]   5.99M  --.-KB/s    in 0.1s

2023-12-22 16:24:50 (48.9 MB/s) - ‘poquad-dev.json’ saved [6286317/6286317]

--2023-12-22 16:24:50--  https://huggingface.co/datasets/clarin-pl/poquad/resolve/main/poquad-train.json
Resolving huggingface.co (huggingface.co)... 18.244.202.60, 18.244.202.73, 18.244.202.68, ...
Connecting to huggingface.co (huggingface.co)|18.244.202.60|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://cdn-lfs.huggingface.co/repos/18/de/18ded45e8046dd5f58b7365947f5a4298433a0e7710248308670e8cf26059c20/b1ac3acabb49fedb7bb7db0de0690ddb22585d6419321589cc1bb0a8068a4ff9?response-content-disposition=attachment%3B+filename*%3DUTF-8%27%27poquad-train.json%3B+filename%3D%22poquad-train.json%22%3B&response-content-type=application%2Fjson&Expires=1703521478&Policy=eyJTdGF0ZW1lbnQiOlt7IkNvbmRpdGlvbiI6eyJEYXRlTGVzc1RoYW4iOnsiQVdTOkVwb2NoVGltZSI6MTcwMzUyMTQ3OH19LCJSZXNvdXJjZSI6Imh0dHBzOi8vY2RuLWxmcy5odWdnaW5nZmFjZS5jby9yZXBvcy8xOC9kZS8xOGRlZDQ1ZTgwNDZkZDVmNThiNzM2NTk0N2Y1YTQyOTg0MzNhMGU3NzEwMjQ4MzA4NjcwZThjZjI2MDU5YzIwL2IxYWMzYWNhYmI0OWZlZGI3YmI3ZGIwZGUwNjkwZGRiMjI1ODVkNjQxOTMyMTU4OWNjMWJiMGE4MDY4YTRmZjk%7EcmVzcG9uc2UtY29udGVudC1kaXNwb3NpdGlvbj0qJnJlc3BvbnNlLWNvbnRlbnQtdHlwZT0qIn1dfQ__&Signature=OKqh%7EVyodhZ%7E3pKq6oIBD-Vq5TJpItnEjKefsDabuYjpsDrvhLNxMJOEaz9xvrshbZbRfsl1xyYnbSxxGOKd63URaVofEluIoqDwpPSMZJu3oky9Bfzp1x17XgdX7OF-qcSJQZOf1TMcFedHsRc2O5rG3cC4Oz7Stx7JTDBQsSzimAlPFRbFIt7m9hntdU8a8ATPzoelwqNrbEsEdlrTErxQmCM0NAr2PtsrQZeA-oVoDBD02fxewdSs4v3PiRAs-t-QOYndPf4aVoB1hX4XgKWVZtcO8yeWbLopIasu0Z2gah1XreTS9F3ql0EUgzwb2mtboBWB2o28wSToFrA1rg__&Key-Pair-Id=KVTP0A1DKRTAX [following]
--2023-12-22 16:24:50--  https://cdn-lfs.huggingface.co/repos/18/de/18ded45e8046dd5f58b7365947f5a4298433a0e7710248308670e8cf26059c20/b1ac3acabb49fedb7bb7db0de0690ddb22585d6419321589cc1bb0a8068a4ff9?response-content-disposition=attachment%3B+filename*%3DUTF-8%27%27poquad-train.json%3B+filename%3D%22poquad-train.json%22%3B&response-content-type=application%2Fjson&Expires=1703521478&Policy=eyJTdGF0ZW1lbnQiOlt7IkNvbmRpdGlvbiI6eyJEYXRlTGVzc1RoYW4iOnsiQVdTOkVwb2NoVGltZSI6MTcwMzUyMTQ3OH19LCJSZXNvdXJjZSI6Imh0dHBzOi8vY2RuLWxmcy5odWdnaW5nZmFjZS5jby9yZXBvcy8xOC9kZS8xOGRlZDQ1ZTgwNDZkZDVmNThiNzM2NTk0N2Y1YTQyOTg0MzNhMGU3NzEwMjQ4MzA4NjcwZThjZjI2MDU5YzIwL2IxYWMzYWNhYmI0OWZlZGI3YmI3ZGIwZGUwNjkwZGRiMjI1ODVkNjQxOTMyMTU4OWNjMWJiMGE4MDY4YTRmZjk%7EcmVzcG9uc2UtY29udGVudC1kaXNwb3NpdGlvbj0qJnJlc3BvbnNlLWNvbnRlbnQtdHlwZT0qIn1dfQ__&Signature=OKqh%7EVyodhZ%7E3pKq6oIBD-Vq5TJpItnEjKefsDabuYjpsDrvhLNxMJOEaz9xvrshbZbRfsl1xyYnbSxxGOKd63URaVofEluIoqDwpPSMZJu3oky9Bfzp1x17XgdX7OF-qcSJQZOf1TMcFedHsRc2O5rG3cC4Oz7Stx7JTDBQsSzimAlPFRbFIt7m9hntdU8a8ATPzoelwqNrbEsEdlrTErxQmCM0NAr2PtsrQZeA-oVoDBD02fxewdSs4v3PiRAs-t-QOYndPf4aVoB1hX4XgKWVZtcO8yeWbLopIasu0Z2gah1XreTS9F3ql0EUgzwb2mtboBWB2o28wSToFrA1rg__&Key-Pair-Id=KVTP0A1DKRTAX
Resolving cdn-lfs.huggingface.co (cdn-lfs.huggingface.co)... 18.173.219.5, 18.173.219.80, 18.173.219.24, ...
Connecting to cdn-lfs.huggingface.co (cdn-lfs.huggingface.co)|18.173.219.
```

```
5|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 47183344 (45M) [application/json]
Saving to: ‘poquad-train.json’

poquad-train.json   100%[===================>]  45.00M  72.6MB/s    in 0.6
s

2023-12-22 16:24:51 (72.6 MB/s) - ‘poquad-train.json’ saved [47183344/4718
3344]
```

Dla bezpieczeństwa, jeśli korzystamy z Google drive, to przeniesiemy pliki do naszego dysku:

In [21]:
```
!mkdir gdrive/MyDrive/poquad
!mv poquad-dev.json gdrive/MyDrive/poquad
!mv poquad-train.json gdrive/MyDrive/poquad

!head -30 gdrive/MyDrive/poquad/poquad-dev.json
```

```
mkdir: cannot create directory ‘gdrive/MyDrive/poquad’: File exists
{
  "version": "02-20",
  "data": [
    {
      "id": 9773,
      "title": "Miszna",
      "summary": "Miszna (hebr. משנה miszna „nauczać”, „ustnie przekazywa
ć”, „studiować”, „badać”, od שנה szana „powtarzać”, „różnić się”, „być odm
iennym”; jid. Miszne) – w judaizmie uporządkowany zbiór tekstów ustnego pr
awa uzupełniający Torę (Prawo pisane). Według wierzeń judaizmu stanowi ust
ną, niespisaną część prawa nadanego przez Boga na Synaju, tzw. Torę ustną.
Jest świętym tekstem judaizmu i jest traktowana na równi z Tanach (Biblią
hebrajską). Zbiór był w Izraelu od wieków przekazywany ustnie z pokolenia
na pokolenie, zwiększył swój rozmiar szczególnie w okresie od III w. p.n.
e. do II w. n.e. w wyniku systematycznego uzupełniania komentarzy przez ta
nnaitów, żydowskich nauczycieli prawa ustnego. Miszna została spisana dopi
ero w II–III w. Prace redakcyjne zapoczątkował rabin Akiba ben Josef, a ks
ztałt ostatecznej redakcji tekstu nadał Juda ha-Nasi. Miszna składa się z
6 porządków (hebr.: sedarim), które dzielą się na 63 traktaty, te zaś na r
ozdziały i lekcje. Miszna jest częścią Talmudu i zawiera podstawowe reguły
postępowania i normy prawne judaizmu.",
      "url": "https://pl.wikipedia.org/wiki/Miszna",
      "paragraphs": [
        {
          "context": "Pisma rabiniczne – w tym Miszna – stanowią kompilacj
ę poglądów różnych rabinów na określony temat. Zgodnie z wierzeniami judai
zmu Mojżesz otrzymał od Boga całą Torę, ale w dwóch częściach: jedną część
w formie pisanej, a drugą część w formie ustnej. Miszna – jako Tora ustna
– była traktowana nie tylko jako uzupełnienie Tory spisanej, ale również j
ako jej interpretacja i wyjaśnienie w konkretnych sytuacjach życiowych. Ty
m samym Miszna stanowiąca kodeks Prawa religijnego zaczęła równocześnie sł
użyć za jego ustnie przekazywany podręcznik.",
          "qas": [
            {
              "question": "Czym są pisma rabiniczne?",
              "answers": [
                {
                  "text": "kompilację poglądów różnych rabinów na określon
y temat",
                  "answer_start": 43,
                  "answer_end": 97,
                  "generative_answer": "kompilacją poglądów różnych rabinó
w na określony temat"
                }
              ],
              "is_impossible": false
            },
            {
              "question": "Z ilu komponentów składała się Tora przekazana
Mojżeszowi?",
              "answers": [
                {
                  "text": "dwóch",
                  "answer_start": 172,
```


Struktura pliku odpowiada strukturze danych w zbiorze SQuAD. Dane umieszczone są w kluczu `data` i podzielone na krotki odpowiadające pojedynczym artykułom Wikipedii. W ramach artykułu może być wybranych jeden lub więcej paragrafów, dla

których w kluczu `qas` pojawiają się pytania (`question`), flaga `is_impossible` , wskazujace czy można odpowiedzieć na pytanie oraz odpowiedzi (o ile nie jest ustawiona flaga `is_impossible`). Odpowiedzi może być wiele i składają się one z treści odpowiedzi (`text`) traktowanej jako fragment kontekstu, a także naturalnej odpowiedzi na pytanie (`generative_answer`).

Taki podział może wydawać się dziwny, ale zbiór SQuAD zawiera tylko odpowiedzi pierwszego rodzaju. Wynika to z faktu, że w języku angielskim fragment tekstu będzie często stanowił dobrą odpowiedź na pytanie (oczywiście z wyjątkiem pytań dla których odpowiedź to `tak` lub `nie`).

Natomiast ten drugi typ odpowiedzi jest szczególnie przydatny dla języka polskiego, ponieważ często odpowiedź chcemy syntaktycznie dostosować do pytania, co jest niemożliwe, jeśli odpowiedź wskazywana jest jako fragment kontekstu. W sytuacji, w której odpowiedzi były określane w sposób automatyczny, są one oznaczone jako `plausible_answers` .

Zaczniemy od wczytania danych i wyświetlenia podstawowych statystyk dotyczących ilości artykułów oraz przypisanych do nich pytań.

In [22]:
```
import json

# Adjust for your needs
path = 'gdrive/MyDrive/poquad'

with open(path + "/poquad-train.json") as input:
    train_data = json.loads(input.read())["data"]

print(f"Train data articles: {len(train_data)}")

with open(path +"/poquad-dev.json") as input:
    dev_data = json.loads(input.read())["data"]

print(f"Dev data articles: {len(dev_data)}")

print(f"Train questions: {sum([len(e['paragraphs'][0]['qas']) for e in tr
print(f"Dev questions: {sum([len(e['paragraphs'][0]['qas']) for e in dev_
```

```
Train data articles: 8553
Dev data articles: 1402
Train questions: 41577
Dev questions: 6809
```

Ponieważ w pierwszym problemie chcemy stwierdzić, czy na pytanie można udzielić odpowiedzi na podstawie kontekstu, połączymy wszystkie konteksty w jedną tablicę, aby móc losować z niej dane negatywne, gdyż liczba pytań nie posiadających odpowiedzi jest stosunkowo mała, co prowadziłoby utworzenia niezbalansowanego zbioru.

In [23]:
```
all_contexts = [e["paragraphs"][0]["context"] for e in train_data] + [
    e["paragraphs"][0]["context"] for e in dev_data
]
```

W kolejnym kroku zamieniamy dane w formacie JSON na reprezentację zgodną z

przyjętym założeniem. Chcemy by kontekst oraz pytanie występowały obok siebie i każdy z elementów był sygnalizowany wyrażeniem: `Pytanie:` i `Kontekst:` . Treść klasyfikowanego tekstu przyporządkowujemy do klucza `text` , natomiast klasę do klucza `label` , gdyż takie są oczekiwanie biblioteki Transformer.

Pytania, które mają ustawiną flagę `is_impossible` na `True` trafiają wprost do przekształconego zbioru. Dla pytań, które posiadają odpowiedź, dodatkowo losowany jest jeden kontekst, który stanowi negatywny przykład. Weryfikujemy tylko, czy kontekst ten nie pokrywa się z kontekstem, który przypisany był do pytania. Nie przeprowadzamy bardziej zaawansowanych analiz, które pomogłyby wylkuczyć sytuację, w której inny kontekst również zawiera odpowiedź na pytanie, gdyż prawdopodobieństwo wylosowania takiego kontekstu jest bardzo małe.

Na końcu wyświetlamy statystyki utworzonego zbioru danych.

In [24]:
```
import random


tuples = [[], []]

for idx, dataset in enumerate([train_data, dev_data]):
    for data in dataset:
        context = data["paragraphs"][0]["context"]
        for question_answers in data["paragraphs"][0]["qas"]:
            question = question_answers["question"]
            if question_answers["is_impossible"]:
                tuples[idx].append(
                    {
                        "text": f"Pytanie: {question} Kontekst: {context}
                        "label": 0,
                    }
                )
            else:
                tuples[idx].append(
                    {
                        "text": f"Pytanie: {question} Kontekst: {context}
                        "label": 1,
                    }
                )
                while True:
                    negative_context = random.choice(all_contexts)
                    if negative_context != context:
                        tuples[idx].append(
                            {
                                "text": f"Pytanie: {question} Kontekst: {
                                "label": 0,
                            }
                        )
                        break

train_tuples, dev_tuples = tuples
print(f"Total count in train/dev: {len(train_tuples)}/{len(dev_tuples)}")
print(
    f"Positive count in train/dev: {sum([e['label'] for e in train_tuples
)
```

```
Total count in train/dev: 75605/12372
Positive count in train/dev: 34028/5563
```

Widzimy, że uzyskane zbiory danych cechują się dość dobrym zbalansowaniem.

Dobrą praktyką po wprowadzeniu zmian w zbiorze danych, jest wyświetlenie kilku przykładowych punktów danych, w celu wykrycia ewentualnych błędów, które powstały na etapie konwersji zbioru. Pozwala to uniknąć nieprzyjemnych niespodzianek, np. stworzenie identycznego zbioru danych testowych i treningowych.

In [25]:
```
print(train_tuples[0:1])
print(dev_tuples[0:1])
```

```
[{'text': 'Pytanie: Co było powodem powrócenia konceptu porozumieniu monac
hijskiego? Kontekst: Projekty konfederacji zaczęły się załamywać 5 sierpni
a 1942. Ponownie wróciła kwestia monachijska, co uaktywniło się wymianą li
stów Ripka – Stroński. Natomiast 17 sierpnia 1942 doszło do spotkania E. B
eneša i J. Masaryka z jednej a Wł. Sikorskiego i E. Raczyńskiego z drugiej
strony. Polscy dyplomaci zaproponowali podpisanie układu konfederacyjnego.
W następnym miesiącu, tj. 24 września, strona polska przesłała na ręce J.
Masaryka projekt deklaracji o przyszłej konfederacji obu państw. Strona cz
echosłowacka projekt przyjęła, lecz już w listopadzie 1942 E. Beneš podważ
ył ideę konfederacji. W zamian zaproponowano zawarcie układu sojuszniczego
z Polską na 20 lat (formalnie nastąpiło to 20 listopada 1942).', 'label':
1}]
[{'text': 'Pytanie: Czym są pisma rabiniczne? Kontekst: Pisma rabiniczne –
w tym Miszna – stanowią kompilację poglądów różnych rabinów na określony t
emat. Zgodnie z wierzeniami judaizmu Mojżesz otrzymał od Boga całą Torę, a
le w dwóch częściach: jedną część w formie pisanej, a drugą część w formie
ustnej. Miszna – jako Tora ustna – była traktowana nie tylko jako uzupełni
enie Tory spisanej, ale również jako jej interpretacja i wyjaśnienie w kon
kretnych sytuacjach życiowych. Tym samym Miszna stanowiąca kodeks Prawa re
ligijnego zaczęła równocześnie służyć za jego ustnie przekazywany podręczn
ik.', 'label': 1}]
```

Ponieważ mamy nowe zbiory danych, możemy opakować je w klasy ułatwiające manipulowanie nimi. Ma to szczególne znaczenie w kontekście szybkiej tokenizacji tych danych, czy późniejszego szybkiego wczytywania wcześniej utworzonych zbiorów danych.

W tym celu wykorzystamy bibliotekę `datasets`. Jej kluczowymi klasami są `Dataset` reprezentujący jeden z podzbiorów zbioru danych (np. podzbiór testowy) oraz `DatasetDict`, który łączy wszystkie podzbiory w jeden obiekt, którym możemy manipulować w całości. (Gdyby autorzy udostępnili odpowiedni skrypt ze zbiorem, moglibyśmy wykorzystać tę bibliotekę bez dodatkowej pracy).

Dodatkowo zapiszemy tak utworzony zbiór danych na dysku. Jeśli później chcielibyśmy wykorzystać stworzony zbiór danych, to możemy to zrobić za pomocą komendy `load_dataset`.

In [26]:
```
from datasets import Dataset, DatasetDict

train_dataset = Dataset.from_list(train_tuples)
dev_dataset = Dataset.from_list(dev_tuples)
```

```
datasets = DatasetDict({"train": train_dataset, "dev": dev_dataset})
datasets.save_to_disk(path + "/question-context-classification")
```

```
Saving the dataset (0/1 shards):   0%|          | 0/75605 [00:00<?, ? exam
ples/s]
Saving the dataset (0/1 shards):   0%|          | 0/12372 [00:00<?, ? exam
ples/s]
```

Dane tekstowe przed przekazaniem do modelu wymagają tokenizacji (co widzieliśmy już wcześniej). Efektywne wykonanie tokenizacji na całym zbiorze danych ułatwione jest przez obiekt `DatasetDict`. Definiujemy funkcję `tokenize_function`, która korzystając z załadowanego tokenizera, zamienia tekst na identyfikatory.

W wywołaniu używamy opcji `padding` - uzupełniamy wszystkie teksty do długości najdłuższego tekstu. Dodatkowo, jeśli któryś tekst wykracza poza maksymalną długość obsługiwaną przez model, to jest on przycinany (`truncation=True`).

Tokenizację aplikujemy do zbioru z wykorzystaniem przetwarzania batchowego (`batched=True`), które pozwala na szybsze stokenizowanie dużego zbioru danych.

In [27]:
```
from transformers import AutoTokenizer

pl_tokenizer = AutoTokenizer.from_pretrained("allegro/herbert-base-cased"

def tokenize_function(examples):
    return pl_tokenizer(examples["text"], padding="max_length", truncatio


tokenized_datasets = datasets.map(tokenize_function, batched=True)
tokenized_datasets["train"]
```

```
Map:   0%|          | 0/75605 [00:00<?, ? examples/s]
Map:   0%|          | 0/12372 [00:00<?, ? examples/s]
```

Out[27]:
```
Dataset({
    features: ['text', 'label', 'input_ids', 'token_type_ids', 'attentio
n_mask'],
    num_rows: 75605
})
```

Stokenizowane dane zawierają dodatkowe pola: `input_ids`, `token_type_ids` oraz `attention_mask`. Dla nas najważniejsze jest pole `input_ids`, które zawiera identyfikatory tokenów. Pozostałe dwa pola są ustawione na identyczne wartości (wszystkie tokeny mają ten sam typ, maska atencji zawiera wszystkie niezerowe tokeny), więc nie są one dla nas zbyt interesujące. Zobaczmy pola `text`, `input_ids` oraz `attention_mask` dla pierwszego przykładu:

In [28]:
```
example = tokenized_datasets["train"][0]
print(example["text"])
print(example["input_ids"])
print(example["attention_mask"])
```

Pytanie: Co było powodem powrócenia konceptu porozumieniu monachijskiego?
Kontekst: Projekty konfederacji zaczęły się załamywać 5 sierpnia 1942. Pon
ownie wróciła kwestia monachijska, co uaktywniło się wymianą listów Ripka
– Stroński. Natomiast 17 sierpnia 1942 doszło do spotkania E. Beneša i J.
Masaryka z jednej a Wł. Sikorskiego i E. Raczyńskiego z drugiej strony. Po
lscy dyplomaci zaproponowali podpisanie układu konfederacyjnego. W następn
ym miesiącu, tj. 24 września, strona polska przesłała na ręce J. Masaryka
projekt deklaracji o przyszłej konfederacji obu państw. Strona czechosłowa
cka projekt przyjęła, lecz już w listopadzie 1942 E. Beneš podważył ideę k
onfederacji. W zamian zaproponowano zawarcie układu sojuszniczego z Polską
na 20 lat (formalnie nastąpiło to 20 listopada 1942).
[0, 14142, 1335, 3407, 2404, 14736, 6491, 4081, 6743, 2213, 19824, 25437,
3096, 13875, 1550, 2922, 8413, 1335, 46771, 2152, 17914, 10278, 2022, 1131
4, 37410, 1008, 4983, 19240, 1899, 38382, 14919, 9091, 25437, 3096, 18290,
1947, 2249, 89, 12539, 2742, 2135, 2022, 45310, 8404, 6242, 8221, 1680, 77
01, 2547, 1899, 5016, 2571, 4983, 19240, 5073, 2041, 4261, 1039, 1899, 132
87, 254, 1011, 1009, 1071, 1899, 2306, 7634, 40838, 1046, 4192, 1011, 59,
1032, 1899, 25496, 1009, 1039, 1899, 2710, 11553, 1046, 3885, 3441, 1899,
24175, 11467, 2057, 35267, 26991, 10374, 2152, 11098, 1990, 4206, 1899, 10
49, 13360, 11794, 1947, 11171, 1899, 2902, 4464, 1947, 11287, 7677, 11610,
2158, 1998, 6428, 1071, 1899, 2306, 7634, 40838, 4555, 17695, 1007, 24584,
2152, 17914, 5351, 8373, 1899, 24649, 2052, 2014, 21425, 3056, 4555, 1624
8, 1947, 4269, 2267, 1019, 11738, 19240, 1039, 1899, 13287, 1107, 33851, 4
561, 28060, 2152, 17914, 1899, 1049, 11721, 34366, 36004, 10374, 20003, 53
11, 1046, 11695, 1998, 2440, 2460, 1341, 19046, 14608, 2063, 2440, 5252, 1
9240, 1940, 1899, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1

Możem też sprawdzić, jak został stokenizowany pierwszy przykład:

```
In [29]: print("|".join(pl_tokenizer.convert_ids_to_tokens(list(example["input_ids
```

```
<s>|Pytanie</w>|:</w>|Co</w>|było</w>|powodem</w>|powró|cenia</w>|koncep|t
u</w>|porozumieniu</w>|mona|chi|jskiego</w>|?</w>|Kon|tekst</w>|:</w>|Proj
ekty</w>|kon|federacji</w>|zaczęły</w>|się</w>|zała|mywać</w>|5</w>|sierpn
ia</w>|1942</w>|.</w>|Ponownie</w>|wróciła</w>|kwestia</w>|mona|chi|jska</
w>|,</w>|co</w>|u|akty|wni|ło</w>|się</w>|wymianą</w>|listów</w>|Ri|pka</w
>|—</w>|Stro|ński</w>|.</w>|Natomiast</w>|17</w>|sierpnia</w>|1942</w>|dos
zło</w>|do</w>|spotkania</w>|E</w>|.</w>|Bene|š|a</w>|i</w>|J</w>|.</w>|M
a|sar|yka</w>|z</w>|jednej</w>|a</w>|W|ł</w>|.</w>|Sikorskiego</w>|i</w>|E
</w>|.</w>|Ra|czyńskiego</w>|z</w>|drugiej</w>|strony</w>|.</w>|Polscy</w
>|dyploma|ci</w>|zaproponowali</w>|podpisanie</w>|układu</w>|kon|fede|ra|c
yjnego</w>|.</w>|W</w>|następnym</w>|miesiącu</w>|,</w>|tj</w>|.</w>|24</w
>|września</w>|,</w>|strona</w>|polska</w>|przesł|ała</w>|na</w>|ręce</w>|
J</w>|.</w>|Ma|sar|yka</w>|projekt</w>|deklaracji</w>|o</w>|przyszłej</w>|
kon|federacji</w>|obu</w>|państw</w>|.</w>|Strona</w>|cze|ch|osłowa|cka</w
>|projekt</w>|przyjęła</w>|,</w>|lecz</w>|już</w>|w</w>|listopadzie</w>|19
42</w>|E</w>|.</w>|Bene|š</w>|podwa|żył</w>|ideę</w>|kon|federacji</w>|.</
w>|W</w>|zamian</w>|zaproponowano</w>|zawarcie</w>|układu</w>|sojusz|nicze
go</w>|z</w>|Polską</w>|na</w>|20</w>|lat</w>|(</w>|formalnie</w>|nastąpił
o</w>|to</w>|20</w>|listopada</w>|1942</w>|)</w>|.</w>|</s>|<pad>|<pad>|<p
ad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad>|<pad
>|<pad>|<pad>|<pad>|<pad>
```

Widzimy, że wyrazy podzielone są sensownie, a na końcu tekstu pojawiają się tokeny wypełnienia (PAD). Oznacza to, że zdanie zostało poprawnie skonwertowane.

Możemy sprawdzić, że liczba tokenów w polu `inut_ids` , które są różne od tokenu wypełnienia (`[PAD] = 1`) oraz maska atencji, mają tę samą długość:

In [30]:
```
print(len([e for e in example["input_ids"] if e != 1]))
print(len([e for e in example["attention_mask"] if e == 1]))
```

```
169
169
```

Mając pewność, że przygotowane przez nas dane są prawidłowe, możemy przystąpić do procesu uczenia modelu.

Trening z użyciem transformersów

Biblioteka Transformes pozwala na załadowanie tego samego modelu dostosowanego do różnych zadań. Wcześniej używaliśmy modelu HerBERT do predykcji brakującego wyrazu. Teraz załadujemy ten sam model, ale z inną "głową". Zostanie użyta warstwa, która pozwala na klasyfikację całego tekstu do jednej z n-klas. Wystarczy podmienić klasę, za pomocą której ładujemy model na `AutoModelForSequenceClassification` :

In [31]:
```
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(
    "allegro/herbert-base-cased", num_labels=2
)

model
```

```
Some weights of BertForSequenceClassification were not initialized from th
e model checkpoint at allegro/herbert-base-cased and are newly initialize
d: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to u
se it for predictions and inference.
```

```
Out[31]: BertForSequenceClassification(
           (bert): BertModel(
             (embeddings): BertEmbeddings(
               (word_embeddings): Embedding(50000, 768, padding_idx=1)
               (position_embeddings): Embedding(514, 768)
               (token_type_embeddings): Embedding(2, 768)
               (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
               (dropout): Dropout(p=0.1, inplace=False)
             )
             (encoder): BertEncoder(
               (layer): ModuleList(
                 (0-11): 12 x BertLayer(
                   (attention): BertAttention(
                     (self): BertSelfAttention(
                       (query): Linear(in_features=768, out_features=768, bias=Tr
         ue)
                       (key): Linear(in_features=768, out_features=768, bias=Tru
         e)
                       (value): Linear(in_features=768, out_features=768, bias=Tr
         ue)
                       (dropout): Dropout(p=0.1, inplace=False)
                     )
                     (output): BertSelfOutput(
                       (dense): Linear(in_features=768, out_features=768, bias=Tr
         ue)
                       (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affi
         ne=True)
                       (dropout): Dropout(p=0.1, inplace=False)
                     )
                   )
                   (intermediate): BertIntermediate(
                     (dense): Linear(in_features=768, out_features=3072, bias=Tru
         e)
                     (intermediate_act_fn): GELUActivation()
                   )
                   (output): BertOutput(
                     (dense): Linear(in_features=3072, out_features=768, bias=Tru
         e)
                     (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine
         =True)
                     (dropout): Dropout(p=0.1, inplace=False)
                   )
                 )
               )
             )
             (pooler): BertPooler(
               (dense): Linear(in_features=768, out_features=768, bias=True)
               (activation): Tanh()
             )
           )
           (dropout): Dropout(p=0.1, inplace=False)
           (classifier): Linear(in_features=768, out_features=2, bias=True)
         )
```

Komunikat diagnostyczny, który pojawia się przy ładowaniu modelu jest zgodny z naszymi oczekiwaniami. Model HerBERT był trenowany do predykcji tokenów, a nie klasyfikacji tekstu. Dlatego też ostatnia warstwa (`classifier.weight` oraz `classifier.bias`) jest inicjowana losowo. Wagi zostaną ustalone w trakcie

procesu fine-tuningu modelu.

Jeśli porównamy wersje modeli załadowane za pomocą różnych klas, to zauważymy, że różnią się one tylko na samym końcu. Jest to zgodne z założeniami procesu pre-treningu i fine-tuningu. W pierwszy etapie model uczy się zależności w języku, korzystając z zadania maskowanego modelowania języka (Masked Language Modeling). W drugim etapie model dostosowywane jest do konkretnego zadania, np. klasyfikacji binarnej tekstu.

Korzystanie z biblioteki Transformers uwalnia nas od manualnego definiowania pętli uczącej, czy wywoływania algorytmu wstecznej propagacji błędu. Trening realizowany jest z wykorzystaniem klasy `Trainer` (i jej specjlizacji). Argumenty treningu określane są natomiast w klasie `TrainingArguments`. Klasy te są bardzo dobrze udokumentowane, więc nie będziemy omawiać wszystkich możliwych opcji.

Najważniejsze opcje są następujące:

- `output_dir` - katalog do którego zapisujemy wyniki,
- `do_train` - wymagamy aby przeprowadzony był trening,
- `do_eval` - wymagamy aby przeprowadzona była ewaluacja modelu,
- `evaluation_strategy` - określenie momentu, w którym realizowana jest ewaluacja,
- `evaluation_steps` - określenie co ile kroków (krok = przetworzenie 1 batcha) ma być realizowana ewaluacja,
- `per_device_train/evaluation_batch_size` - rozmiar batcha w trakcie treningu/ewaluacji,
- `learning_rate` - szybkość uczenia,
- `num_train_epochs` - liczba epok uczenia,
- `logging` ... - parametry logowania postępów uczenia,
- `save_strategy` - jak często należy zapisywać wytrenowany model,
- `fp16/bf16` - użycie arytmetyki o zmniejszonej dokładności, przyspieszającej proces uczenia. **UWAGA**: użycie niekompatybilnej arytmetyki skutkuje niemożnością nauczenia modelu, co jednak nie daje żadnych innych błędów lub komunikatów ostrzegawczych.

In [32]:
```
from transformers import TrainingArguments
import numpy as np

arguments = TrainingArguments(
    output_dir=path + "/output",
    do_train=True,
    do_eval=True,
    evaluation_strategy="steps",
    eval_steps=300,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=5e-05,
    num_train_epochs=1,
    logging_first_step=True,
    logging_strategy="steps",
    logging_steps=50,
```

```
    save_strategy="epoch",
    fp16=True,
)
```

W trakcie treningu będziemy chcieli zobaczyć, czy model poprawnie radzi sobie z postawionym mu problemem. Najlepszym sposobem na podglądanie tego procesu jest obserwowanie wykresów. Model może raportować szereg metryk, ale najważniejsze dla nas będą następujące wartości:

- wartość funkcji straty na danych treningowych - jeślie nie spada w trakcie uczenia, znaczy to, że nasz model nie jest poprawnie skonstruowany lub dane uczące są niepoprawne,
- wartość jednej lub wielu metryk uzyskiwanych na zbiorze walidacyjnym - możemy śledzić wartość funkcji straty na zbiorze ewaluacyjnym, ale warto również wyświetlać metryki, które da się łatwiej zinterpretować; dla klasyfikacji zbalansowanego zbioru danych może to być dokładność (`accuracy`).

Biblioteka Transformers pozwala w zasadzie na wykorzystanie dowolnej metryki, ale szczególnie dobrze współpracuje z metrykami zdefiniowanymi w bibliotece `evaluate` (również autorstwa Huggingface).

Wykorzystanie metryki wymaga od nas zdefiniowania metody, która akceptuje batch danych, który zawieraja predykcje (wektory zwrócone na wyjściu modelu) oraz referencyjne wartości - wartości przechowywane w kluczu `label` . Przed obliczeniem metryki konieczne jest "odcyfrowanie" zwróconych wartości. W przypadku klasyfikacji oznacza to po prostu wybranie najbardziej prawodopodobnej klasy i porównanie jej z klasą referencyjną.

Użycie konkretnej metryki realizowane jest za pomocą wywołania `metric.compute` , która akceptuje predykcje (`predictions`) oraz wartości referencyjne (`references`).

In [33]:
```
import evaluate

metric = evaluate.load("accuracy")


def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=1)
    return metric.compute(predictions=predictions, references=labels)
```

```
Downloading builder script:   0%|          | 0.00/4.20k [00:00<?, ?B/s]
```

Ostatnim krokiem w procesie treningu jest stworzenie obiektu klasy `Trainer` . Akceptuje ona m.in. model, który wykorzystywany jest w treningu, przygotowane argumenty treningu, zbiory do treningu, ewaluacji, czy testowania oraz wcześniej określoną metodę do obliczania metryki na danych ewaluacyjnych.

W przetwarzaniu jezyka naturalnego dominującym podejściem jest obecnie rozdzielenie procesu treningu na dwa etapy: pre-treining oraz fine-tuning. W pierwszym etapie model trenowany jest w reżimie self-supervised learning (SSL).

Wybierane jest zadanie związane najczęściej z modelowaniem języka - może to być kauzalne lub maskowane modelowanie języka.

W *kauzalnym modelowaniu języka* model językowy, na podstawie poprzedzających wyrazów określa prawdopodobieństwo wystąpienia kolejnego wyrazu. W *maskowanym modelowaniu języka* model językowy odgaduje w tekście część wyrazów, która została z niego usunięta.

W obu przypadkach dane, na których trenowany jest model nie wymagają ręcznego oznakowania (tagowaina). Wystarczy jedynie posiadać duży korpus danych językowych, aby wytrenować model, który dobrze radzi sobie z jednym z tych zadań. Model tego rodzaju był pokazany na początku laboratorium.

W drugim etapie - fine-tuningu (dostrajaniu modelu) - następuje modyfikacja parametrów modelu, w celu rozwiązania konkretnego zadania. W naszym przypadku pierwszym zadaniem tego rodzaju jest klasyfikacja. Dostroimy zatem model `herbert-base-cased` do zadania klasyfikacji par: pytanie - kontekst.

Wykorzystamy wcześniej utworzone zbiory danych i dodatkowo zmienimy kolejność danych, tak aby uniknąć potencjalnego problemu z korelacją danych w ramach batcha. Wykorzystujemy do tego wywołanie `shuffle` .

In [34]:
```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=arguments,
    train_dataset=tokenized_datasets["train"].shuffle(seed=42),
    eval_dataset=tokenized_datasets["dev"].shuffle(seed=42),
    compute_metrics=compute_metrics,
)
```

Zanim uruchomimy trening, załadujemy jeszcze moduł TensorBoard. Nie jest to krok niezbędy. TensorBoard to biblioteka, która pozwala na wyświetlanie w trakcie procesu trening wartości, które wskazują nam, czy model trenuje się poprawnie. W naszym przypadku będzie to `loss` na danych treningowych, `loss` na danych ewaluacyjnych oraz wartość metryki `accuracy` , którą zdefiniowaliśmy wcześniej. Wywołanie tej komórki na początku nie da żadnego efektu, ale można ją odświeżać, za pomocą ikony w menu TensorBoard (ewentualnie włączyć automatyczne odświeżanie). Wtedy w miarę upływu treningu będziemy mieli podgląd, na przebieg procesu oraz osiągane wartości interesujących nas parametrów.

Warto zauważyć, że istenieje szereg innych narzędzi do monitorowania eksperymentów z treningiem sieci. Wśród nich dużą popularnością cieszą się WanDB oraz Neptune.AI. Ich zaletą jest m.in. to, że możemy łatwo archiwizować przeprowadzone eksperymenty, porównywać je ze sobą, analizować wpływ hiperparametrów na uzyskane wyniki, itp.

In [35]:
```
%load_ext tensorboard
%tensorboard --logdir gdrive/MyDrive/poquad/output/runs
```

Uruchomienie procesu treningu jest już bardzo proste, po tym jak przygotowaliśmy wszystkie niezbędne szczegóły. Wystarczy wywołać metodę `trainer.train()` . Warto mieć na uwadze, że proces ten będzie jednak długotrwały - jedna epoka treningu na przygotowanych danych będzie trwała ponad 1 godzinę. Na szczęście, dzięki ustawieniu ewaluacji co 300 kroków, będziemy mogli obserwować jak model radzie sobie z postawionym przed nim problemem na danych ewaluacyjnych.

In [36]:
```
# trainer.train()
# model.save_pretrained(path + "/pretrained-herbert-base-cased")

model = AutoModelForSequenceClassification.from_pretrained(path + "/pretr
```

Zadanie 3 (1 punkt)

Wybierz losową stronę z Wikipedii i skopiuj fragment tekstu do Notebook. Zadaj 3 pytania, na które można udzielić odpowiedź na podstawie tego fragmentu tekstu oraz 3 pytania, na które nie można udzielić odpowiedzi. Oceń jakość predykcji udzielanych przez model.

In [37]:
```
from transformers import TextClassificationPipeline


article = """
Gerlach (główny egzonim) lub Gierlach (egzonim wariantowy), dawniej też
Garłuch (słow. Gerlachovský štít, Gerlachovka, niem. Gerlsdorfer Spitze,
Gerlach, węg. Gerlachfalvi-csúcs), 2655 m n.p.m. – najwyższy szczyt Tatr
oraz całych Karpat i jednocześnie Słowacji, położony w bocznej grani
Tatr Wysokich. Szczyt ten należy do Korony Europy oraz Wielkiej Korony Ta

Masyw Gerlacha składa się z kilku wierzchołków. Grań rozpoczyna się od st
północno-zachodniej wierzchołkiem zwornikowym z granią główną – Zadnim Ge
(Zadný Gerlach, 2616 m), którego zbocza obniżają się do Przełęczy Tetmaje
(Gerlachovské sedlo), za którą wznosi się główny wierzchołek (2655 m).
Dalej w grani znajdują się kolejno Wyżnie Gerlachowskie Wrótka, wierzchoł
Pośredniego Gerlacha, Pośrednie Gerlachowskie Wrótka, Gerlachowska Czuba
i Niżnie Gerlachowskie Wrótka. Grzbiet kończy się południowo-wschodnim sz
Małego Gerlacha (Kotlový štít, 2601 m). Z wierzchołka Małego Gerlacha odc
dwa boczne ramiona obejmujące Gerlachowski Kocioł (Gerlachovský kotol).
"""

def predict(question: str, context: str, model, tokenizer):
    question_with_context =f"Pytanie: {question} Kontekst: {context}"
    pipeline = TextClassificationPipeline(
        model=model,
        tokenizer=tokenizer,
        device=0
    )

    return pipeline(question_with_context)

questions = [
```

```
    {
        "value": "Jak się nazywa najwyższy szczyt Tatr?",
        "is_possible": True
    },
    {
        "value": "Jaką wysokość ma Gerlach?",
        "is_possible": True
    },
    {
        "value": "Jak się nazywa przełęcz na Gerlachu nazwana na cześć" +
            "polskiego poety?",
        "is_possible": True
    },
    {
        "value": "Czy Mont Blanc to najwyższy szczyt w Alpach?",
        "is_possible": False
    },
    {
        "value": "Czy pomidory można uprawiać w Polsce?",
        "is_possible": False
    },
    {
        "value": "Jak się nazywa śmiertelna choroba przenoszona przez" +
            "zwierzęta, między innymi psy?",
        "is_possible": False
    },
]

for question in questions:
    is_possible = question["is_possible"]
    question    = question["value"]
    pred        = predict(question, article, model, pl_tokenizer)
    pred_label  = pred[0]["label"]
    pred_score  = pred[0]["score"]

    if pred_label == "LABEL_1" and is_possible:
        print(f"Prediction is possible with confidence {pred_score:.1%}."
    elif pred_label == "LABEL_1" and not is_possible:
        print(f"Prediction is possible with confidence {pred_score:.1%},
            "however it cannot be possible to predict the sentence.")
    elif pred_label == "LABEL_0" and not is_possible:
        print(f"Prediction is impossible with confidence {pred_score:.1%}
    elif pred_label == "LABEL_0" and is_possible:
        print(f"Prediction is impossible with confidence {pred_score:.1%}
            "however it cannot be possible to predict the sentence.")
```

```
Prediction is possible with confidence 74.5%.
Prediction is possible with confidence 91.8%.
Prediction is impossible with confidence 86.3%, however it cannot be possi
ble to predict the sentence.
Prediction is possible with confidence 73.7%, however it cannot be possibl
e to predict the sentence.
Prediction is impossible with confidence 100.0%.
Prediction is impossible with confidence 100.0%.
```

Komentarz

Prawie wszystkie predykcje zostały poprawnie oznaczone z wysokimi wynikami, oprócz, paradoksalnie, najłatwiejszego pytania w tym przypadku. Prawdopodobieństwo odpowiedzenia nie jest bardzo niskie, ale jest zauważalnie

mniejsze. Model potrafi źle zinterpretować o najwyższy szczyt Alp z pewnością w przedziałach ok. 55 - 77% dla zmniejszonej dokładności arytmetyki zmiennoprzecinkowej.

Odpowiadanie na pytania

Drugim problemem, którym zajmie się w tym laboratorium jest odpowiadanie na pytania. Zmierzymy się z wariantem tego problemu, w którym model sam formułuje odpowiedź, na podstawie pytania i kontekstu, w których znajduje się odpowiedź na pytanie (w przeciwieństwie do wariantu, w którym model wskazuje lokalizację odpowiedzi na pytanie).

Zadanie 4 (1 punkt)

Rozpocznij od przygotowania danych. Wybierzem tylko te pytania, które posiadają odpowiedź (`is_impossible=False`). Uwzględnij zarówno pytania *pewne* (pole `answers`) jak i *prawdopodobne* (pole `plausible_answers`). Wynikowy zbiór danych powinien mieć identyczną strukturę, jak w przypadku zadania z klasyfikacją, ale etykiety zamiast wartości 0 i 1, powinny zawierać odpowiedź na pytanie, a sama nazwa etykiety powinna być zmieniona z `label` na `labels` , w celu odzwierciedlenia faktu, że teraz zwracane jest wiele etykiet.

Wyświetl liczbę danych (par: pytanie - odpowiedź) w zbiorze treningowym i zbiorze ewaluacyjnym.

Opakuj również zbiory w klasy z biblioteki `datasets` i zapisz je na dysku.

In [38]:
```
import random


tuples = [[], []]

for idx, dataset in enumerate([train_data, dev_data]):
    for data in dataset:
        context = data["paragraphs"][0]["context"]
        for question_answers in data["paragraphs"][0]["qas"]:
            question = question_answers["question"]
            if question_answers["is_impossible"]:
                continue

            for a in question_answers["answers"]:
                tuples[idx].append(
                    {
                        "text": f"Pytanie: {question} Kontekst: {context}
                        "labels": a["generative_answer"],
                    }
                )

            for a in question_answers.get("plausible_anwsers", []):
                tuples[idx].append(
                    {
```

```
                        "text": f"Pytanie: {question} Kontekst: {context}",
                        "labels": a["generative_answer"],
                    }
                )

train_tuples, dev_tuples = tuples
train_dataset = Dataset.from_list(train_tuples)
dev_dataset   = Dataset.from_list(dev_tuples)
datasets = DatasetDict({"train": train_dataset, "dev": dev_dataset})
datasets.save_to_disk(path + "/question-answering")
```

```
Saving the dataset (0/1 shards):   0%|          | 0/34028 [00:00<?, ? exam
ples/s]
Saving the dataset (0/1 shards):   0%|          | 0/5563 [00:00<?, ? examp
les/s]
```

Zanim przejdziemy do dalszej części, sprawdźmy, czy dane zostały poprawnie utworzone. Zweryfikujmy przede wszystkim, czy klucze `text` oraz `label` zawieraja odpowiednie wartości:

In [39]:
```
print(datasets["train"][0]["text"])
print(datasets["train"][0]["labels"])
print(datasets["dev"][0]["text"])
print(datasets["dev"][0]["labels"])
```

```
Pytanie: Co było powodem powrócenia konceptu porozumieniu monachijskiego?
Kontekst: Projekty konfederacji zaczęły się załamywać 5 sierpnia 1942. Pon
ownie wróciła kwestia monachijska, co uaktywniło się wymianą listów Ripka
– Stroński. Natomiast 17 sierpnia 1942 doszło do spotkania E. Beneša i J.
Masaryka z jednej a Wł. Sikorskiego i E. Raczyńskiego z drugiej strony. Po
lscy dyplomaci zaproponowali podpisanie układu konfederacyjnego. W następn
ym miesiącu, tj. 24 września, strona polska przesłała na ręce J. Masaryka
projekt deklaracji o przyszłej konfederacji obu państw. Strona czechosłowa
cka projekt przyjęła, lecz już w listopadzie 1942 E. Beneš podważył ideę k
onfederacji. W zamian zaproponowano zawarcie układu sojuszniczego z Polską
na 20 lat (formalnie nastąpiło to 20 listopada 1942).
wymiana listów Ripka – Stroński
Pytanie: Czym są pisma rabiniczne? Kontekst: Pisma rabiniczne – w tym Misz
na – stanowią kompilację poglądów różnych rabinów na określony temat. Zgod
nie z wierzeniami judaizmu Mojżesz otrzymał od Boga całą Torę, ale w dwóch
częściach: jedną część w formie pisanej, a drugą część w formie ustnej. Mi
szna – jako Tora ustna – była traktowana nie tylko jako uzupełnienie Tory
spisanej, ale również jako jej interpretacja i wyjaśnienie w konkretnych s
ytuacjach życiowych. Tym samym Miszna stanowiąca kodeks Prawa religijnego
zaczęła równocześnie służyć za jego ustnie przekazywany podręcznik.
kompilacją poglądów różnych rabinów na określony temat
```

Tokenizacja danych dla problemu odpowiadania na pytania jest nieco bardziej problematyczna. W pierwszej kolejności trzeba wziąć pod uwagę, że dane wynikowe (etykiety), też muszą podlegać tokenizacji. Realizowane jest to poprzez wywołanie tokenizera, z opcją `text_target` ustawioną na łańcuch, który ma być stokenizowany.

Ponadto wcześniej nie przejmowaliśmy się za bardzo tym, czy wykorzystywany model obsługuje teksty o założonej długości. Teraz jednak ma to duże znaczenie. Jeśli użyjemy modelu, który nie jest w stanie wygenerować odpowiedzi o oczekiwanej długości, to nie możemy oczekiwać, że model ten będzie dawał dobre rezultaty dla

danych w zbiorze treningowym i testowym.

W pierwszej kolejności dokonamy więc tokenizacji bez ograniczeń co do długości tekstu. Ponadto, stokenizowane odpowiedzi przypiszemy do klucza `label`. Do tokenizacji użyjemy tokenizera stowarzyszonego z modelem `allegro/plt5-base`.

In [40]:
```
from transformers import AutoTokenizer

plt5_tokenizer = AutoTokenizer.from_pretrained("allegro/plt5-base")


def preprocess_function(examples):
    model_inputs = plt5_tokenizer(examples["text"])
    labels = plt5_tokenizer(text_target=examples["labels"])
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs


tokenized_datasets = datasets.map(preprocess_function, batched=True)
```

```
tokenizer_config.json:   0%|          | 0.00/141 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/658 [00:00<?, ?B/s]
spiece.model:   0%|          | 0.00/1.12M [00:00<?, ?B/s]
special_tokens_map.json:   0%|          | 0.00/65.0 [00:00<?, ?B/s]
You are using the default legacy behaviour of the <class 'transformers.mod
els.t5.tokenization_t5.T5Tokenizer'>. This is expected, and simply means t
hat the `legacy` (previous) behavior will be used so nothing changes for y
ou. If you want to use the new behaviour, set `legacy=False`. This should
only be set if you understand what it means, and thouroughly read the reas
on why this was added as explained in https://github.com/huggingface/trans
formers/pull/24565
Map:   0%|          | 0/34028 [00:00<?, ? examples/s]
Map:   0%|          | 0/5563 [00:00<?, ? examples/s]
```

Sprawdźmy jak dane wyglądają po tokenizacji:

In [41]:
```
print(tokenized_datasets["train"][0].keys())
print(tokenized_datasets["train"][0]["input_ids"])
print(tokenized_datasets["train"][0]["labels"])
print(len(tokenized_datasets["train"][0]["input_ids"]))
print(len(tokenized_datasets["train"][0]["labels"]))
example = tokenized_datasets["train"][0]

print("|".join(plt5_tokenizer.convert_ids_to_tokens(list(example["input_i
print("|".join(plt5_tokenizer.convert_ids_to_tokens(list(example["labels"
```

```
dict_keys(['text', 'labels', 'input_ids', 'attention_mask'])
[21584, 291, 639, 402, 11586, 292, 23822, 267, 1269, 8741, 280, 24310, 424
04, 305, 373, 1525, 15643, 291, 2958, 273, 19605, 6869, 271, 298, 2256, 74
65, 394, 540, 2142, 259, 17542, 13760, 10331, 9511, 322, 31220, 261, 358,
348, 267, 7243, 430, 470, 271, 39908, 20622, 2178, 18204, 308, 8439, 2451,
259, 1974, 455, 540, 2142, 1283, 272, 994, 525, 259, 15697, 1978, 267, 26
4, 644, 259, 14988, 19434, 265, 1109, 287, 274, 357, 259, 21308, 264, 525,
259, 35197, 305, 265, 793, 823, 259, 25318, 2750, 4724, 31015, 21207, 416
2, 40335, 18058, 259, 274, 4862, 7030, 261, 5269, 259, 658, 497, 261, 697
1, 1890, 35042, 267, 266, 3260, 644, 259, 14988, 19434, 1187, 20919, 284,
27584, 19605, 1230, 2555, 259, 12531, 7278, 3845, 8726, 10486, 1187, 1067
6, 261, 996, 347, 260, 2548, 2142, 525, 259, 15697, 1978, 309, 27648, 3188
7, 19605, 259, 274, 4931, 36525, 37011, 4162, 10036, 7141, 265, 6340, 266,
465, 346, 269, 3648, 4383, 6704, 294, 465, 567, 2142, 454, 1]
[13862, 20622, 2178, 18204, 308, 8439, 2451, 1]
165
8
_Pytanie|:|_Co|_było|_powodem|_po|wróceni|a|_kon|cept|u|_porozumieniu|_mon
achijski|ego|?|_Kon|tekst|:|_Projekt|y|_konfederacji|_zaczęły|_się|_za|ła
m|ywać|_5|_sierpnia|_1942|.|_Ponownie|_wróciła|_kwestia|_mon|ach|ijska|,|_
co|_u|a|ktyw|ni|ło|_się|_wymianą|_listów|_Ri|pka|_–|_Stro|ński|.|_Natomias
t|_17|_sierpnia|_1942|_doszło|_do|_spotkania|_E|.|_Bene|š|a|_i|_J|.|_Masa|
ryka|_z|_jednej|_a|_W|ł|.|_Sikorskiego|_i|_E|.|_Raczyński|ego|_z|_drugiej|
_strony|.|_Polscy|_dyplom|aci|_zaproponowali|_podpisanie|_układu|_konfeder
ac|yjnego|.|_W|_następnym|_miesiącu|,|_tj|.|_24|_września|,|_strona|_polsk
a|_przesłał|a|_na|_ręce|_J|.|_Masa|ryka|_projekt|_deklaracji|_o|_przyszłe
j|_konfederacji|_obu|_państw|.|_Strona|_cze|cho|słow|acka|_projekt|_przyję
ła|,|_lecz|_już|_w|_listopadzie|_1942|_E|.|_Bene|š|_pod|ważył|_ideę|_konfe
deracji|.|_W|_zamian|_zaproponowano|_zawarcie|_układu|_sojusz|niczego|_z|_
Polską|_na|_20|_lat|_(|form|alnie|_nastąpiło|_to|_20|_listopada|_194
2|).|</s>
_wymiana|_listów|_Ri|pka|_–|_Stro|ński|</s>
```

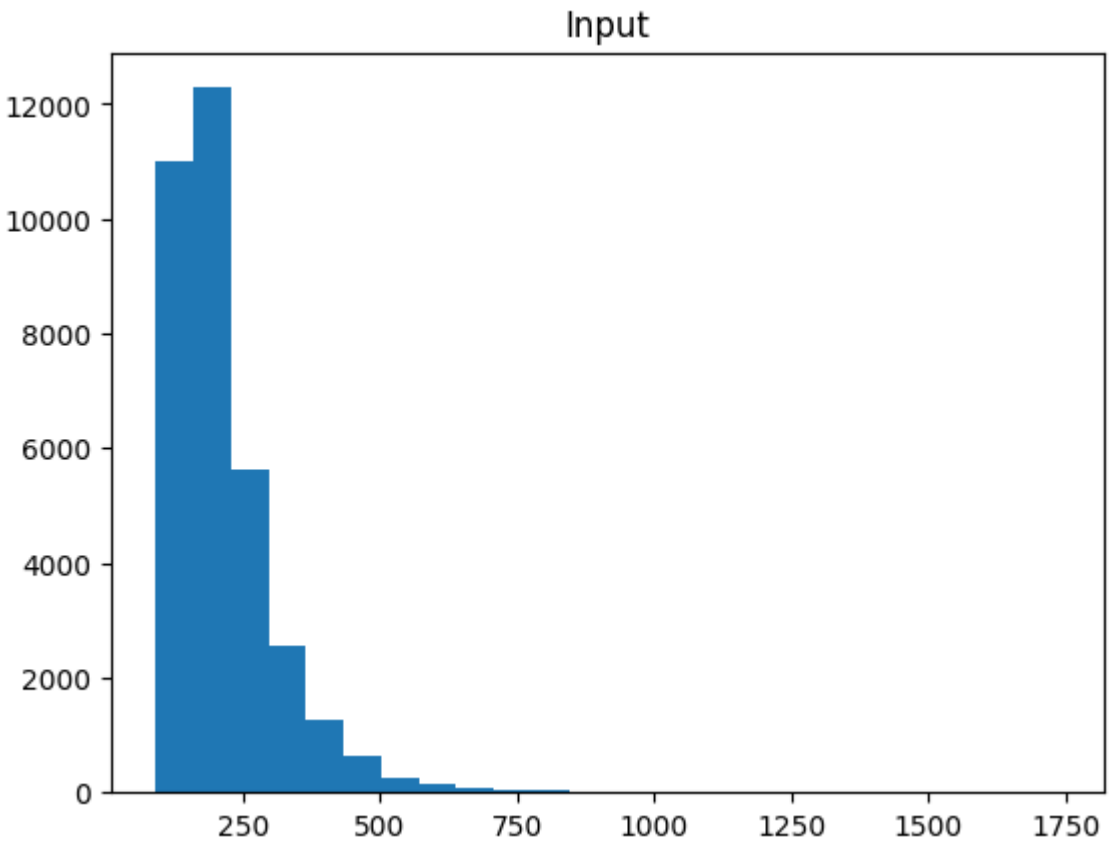
Wykorzystywany przez nas model obsługuje teksty od długości do 512 sub-tokenów (w zasadzie ograniczenie to, w przeciwieństwie do modelu BERT nie wynika z samego modelu, więc teoretycznie moglibyśmy wykorzystywać dłuższe sekwencje, co jednak prowadzi do nadmiernej konsumpcji pamięci). Konieczne jest zatem sprawdzenie, czy w naszych danych nie ma tekstów o większej długości.

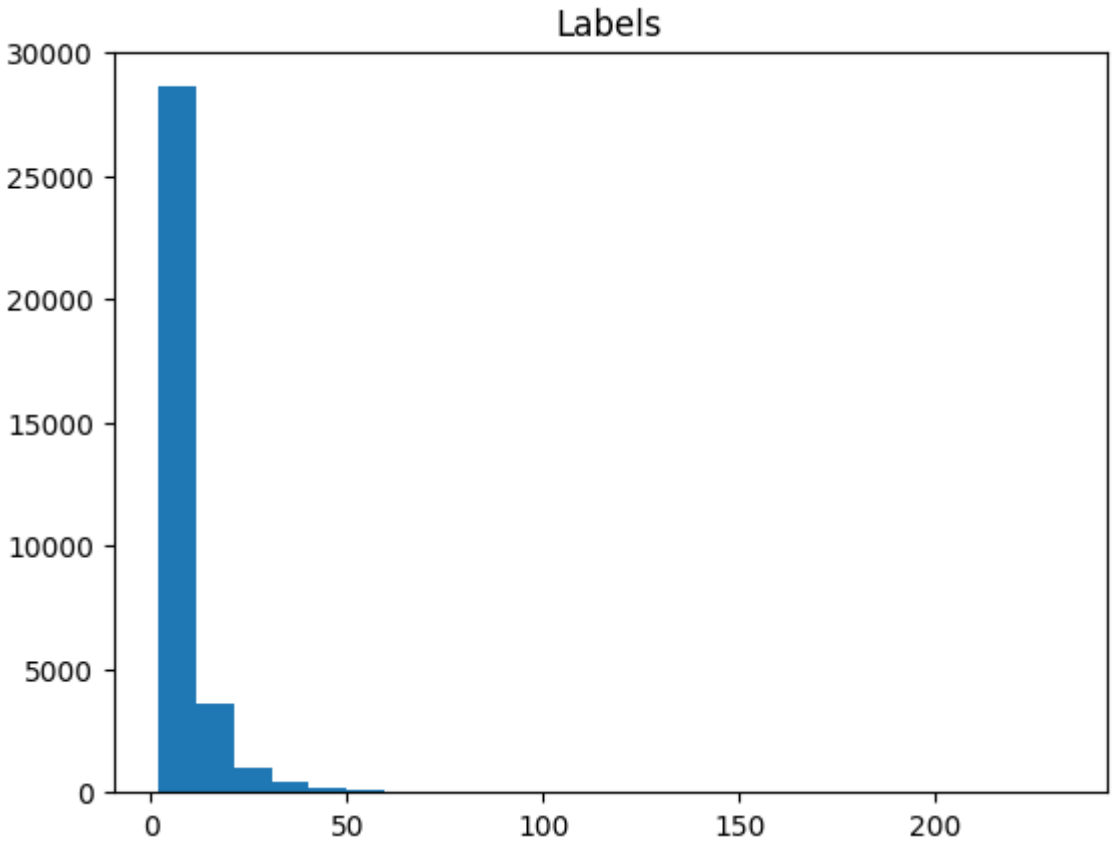
Zadanie 5 (0.5 punkt)

Stwórz histogramy prezentujące rozkład długości (jako liczby tokenów) tekstów wejściowych (`input_ids`) oraz odpowiedzi (`labels`) dla zbioru treningowego. Zinterpretuj otrzymane wyniki.

In [42]:
```python
import matplotlib.pyplot as plt


def histogram(input, title=None, bins=16):
  _, ax = plt.subplots()

  if title:
    ax.set_title(title)
  ax.hist(input, bins=bins)

  plt.show()
```

In [43]:
```
MAX_NO_SUBTOKENS = 2048

counts = {
    "input":  [len(q["input_ids"]) for q in tokenized_datasets["train"]],
    "labels": [len(q["labels"])    for q in tokenized_datasets["train"]]
}

histogram(counts["input"], title="Input", bins=24)
histogram(counts["labels"], title="Labels", bins=24)
```





Absolutna większość danych wejściowych ma długość mniejszą niż 500 tokenów i etykiet mniejszą niż 50.

Przyjmiemy założenie, że teksty wejściowe będą miały maksymalnie 256 tokenów, a większość odpowiedzi jest znacznie krótsza niż maksymalna długość, ograniczmy je do długości 32.

W poniższym kodzie uwzględniamy również fakt, że przy obliczaniu funkcji straty nie interesuje nas wliczanie tokenów wypełnienia (PAD), gdyż ich udział byłby bardzo duży, a nie wpływają one w żaden pozytywny sposób na ocenę poprawności działania modelu.

Konteksty (pytanie + kontekst odpowiedzi) ograniczamy do 256 tokenów, ze względu na ograniczenia pamięciowe (zajętość pamięci dla modelu jest proporcjonalna do kwadratu długości tekstu). Dla kontekstów nie używamy parametru `padding`, ponieważ w trakcie treningu użyjemy modułu, który automatycznie doda padding, tak żeby wszystkie sekewncje miały długość najdłuższego tekstu w ramach paczki (moduł ten to `DataCollatorWithPadding`).

In [44]:
```
def preprocess_function(examples):
    result = plt5_tokenizer(examples["text"], truncation=True, max_length
    targets = plt5_tokenizer(
        examples["labels"], truncation=True, max_length=32, padding=True
    )
    input_ids = [
        [(l if l != plt5_tokenizer.pad_token_id else -100) for l in e]
        for e in targets["input_ids"]
    ]
    result["labels"] = input_ids
```

```
    return result


tokenized_datasets = datasets.map(preprocess_function, batched=True)
```

```
Map:   0%|          | 0/34028 [00:00<?, ? examples/s]
Map:   0%|          | 0/5563 [00:00<?, ? examples/s]
```

Następnie weryfkiujemy, czy przetworzone teksty mają poprawną postać.

In [45]:
```
print(tokenized_datasets["train"][0].keys())
print(tokenized_datasets["train"][0]["input_ids"])
print(tokenized_datasets["train"][0]["labels"])
print(len(tokenized_datasets["train"][0]["input_ids"]))
print(len(tokenized_datasets["train"][0]["labels"]))
```

```
dict_keys(['text', 'labels', 'input_ids', 'attention_mask'])
[21584, 291, 639, 402, 11586, 292, 23822, 267, 1269, 8741, 280, 24310, 424
04, 305, 373, 1525, 15643, 291, 2958, 273, 19605, 6869, 271, 298, 2256, 74
65, 394, 540, 2142, 259, 17542, 13760, 10331, 9511, 322, 31220, 261, 358,
348, 267, 7243, 430, 470, 271, 39908, 20622, 2178, 18204, 308, 8439, 2451,
259, 1974, 455, 540, 2142, 1283, 272, 994, 525, 259, 15697, 1978, 267, 26
4, 644, 259, 14988, 19434, 265, 1109, 287, 274, 357, 259, 21308, 264, 525,
259, 35197, 305, 265, 793, 823, 259, 25318, 2750, 4724, 31015, 21207, 416
2, 40335, 18058, 259, 274, 4862, 7030, 261, 5269, 259, 658, 497, 261, 697
1, 1890, 35042, 267, 266, 3260, 644, 259, 14988, 19434, 1187, 20919, 284,
27584, 19605, 1230, 2555, 259, 12531, 7278, 3845, 8726, 10486, 1187, 1067
6, 261, 996, 347, 260, 2548, 2142, 525, 259, 15697, 1978, 309, 27648, 3188
7, 19605, 259, 274, 4931, 36525, 37011, 4162, 10036, 7141, 265, 6340, 266,
465, 346, 269, 3648, 4383, 6704, 294, 465, 567, 2142, 454, 1]
[13862, 20622, 2178, 18204, 308, 8439, 2451, 1, -100, -100, -100, -100, -1
00, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -10
0, -100, -100, -100, -100, -100, -100, -100]
165
32
```

Dla problemu odpowiadania na pytania potrzebować będziemy innego pre-trenowanego modelu oraz innego przygotowania danych. Jako model bazowy wykrzystamy polski wariant modelu T5 - plT5. Model ten trenowany był w zadaniu *span corruption*, czyli zadani polegającym na usunięciu fragmentu tekstu. Model na wejściu otrzymywał tekst z pominiętymi pewnymi fragmentami, a na wyjściu miał odtwarzać te fragmenty. Oryginalny model T5 dodatkowo pretrenowany był na kilku konkretnych zadaniach z zakresu NLP (w tym odpowiadaniu na pytania). W wariancie plT5 nie przeprowadzono jednak takiego dodatkowego procesu.

Poniżej ładujemy model dla zadania, w którym model generuje tekst na podstawie innego tekstu (tzn. jest to zadanie zamiany tekstu na tekst, po angielsku zwanego też *Sequence-to-Sequence*).

In [46]:
```
from transformers import AutoModelForSeq2SeqLM

model = AutoModelForSeq2SeqLM.from_pretrained("allegro/plt5-base")
```

```
pytorch_model.bin:   0%|          | 0.00/1.10G [00:00<?, ?B/s]
```

Trening modelu QA

Ostatnim krokiem przed uruchomieniem treningu jest zdefiniowanie metryk, wskazujacych jak model radzi sobie z problemem. Wykorzystamy dwie metryki:

- *exact match* - która sprawdza dokładne dopasowanie odpowiedzi do wartości referencyjnej, metryka ta jest bardzo restrykcyjna, ponieważ pojedynczy znak będzie powodował, że wartość będzie niepoprawna,
- *blue score* - metryka uwzględniająca częściowe dopasowanie pomiędzy odpowiedzią a wartością referencyjną, najczęściej używana jest do oceny maszynowego tłumaczenia tekstu, ale może być również przydatna w ocenie wszelkich zadań, w których generowany jest tekst.

Wykorzystujemy bibilotekę `evaluate`, która zawiera definicje obu metryk.

Przy konwersji identyfikatorów tokenów na tekstu zamieniamy również z powroten tokeny o wartości -100 na identyfikatory paddingu. W przeciwnym razie dostaniemy błąd o nieistniejącym identyfikatorze tokenu.

W procesie treningu pokazujemy również różnicę między jedną wygenerowaną oraz prawdziwą odpowiedzią dla zbioru ewaluacyjnego. W ten sposób możemy śledzić co rzeczywiście dzieje się w modelu.

In [47]:
```
from transformers import Seq2SeqTrainer, Seq2SeqTrainingArguments
import numpy as np
import evaluate

exact = evaluate.load("exact_match")
bleu = evaluate.load("bleu")


def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.where(predictions != -100, predictions, plt5_tokeniz
    decoded_preds = plt5_tokenizer.batch_decode(predictions, skip_special
    labels = np.where(labels != -100, labels, plt5_tokenizer.pad_token_id
    decoded_labels = plt5_tokenizer.batch_decode(labels, skip_special_tok
    print("prediction: " + decoded_preds[0])
    print("reference : " + decoded_labels[0])

    result = exact.compute(predictions=decoded_preds, references=decoded_
    result = {**result, **bleu.compute(predictions=decoded_preds, referen
    del result["precisions"]

    prediction_lens = [np.count_nonzero(pred != plt5_tokenizer.pad_token_
    result["gen_len"] = np.mean(prediction_lens)

    return result
```

```
Downloading builder script:   0%|          | 0.00/5.67k [00:00<?, ?B/s]
Downloading builder script:   0%|          | 0.00/5.94k [00:00<?, ?B/s]
Downloading extra modules:   0%|          | 0.00/1.55k [00:00<?, ?B/s]
Downloading extra modules:   0%|          | 0.00/3.34k [00:00<?, ?B/s]
```

Zadanie 6 (0.5 punkty)

Korzystając z klasy Seq2SeqTrainingArguments zdefiniuj następujące parametry trenignu:

- inny katalog z wynikami
- liczba epok: 3
- wielkość paczki: 16
- ewaluacja co 100 kroków,
- szybkość uczenia: 1e-4
- optymalizator: adafactor
- maksymalna długość generowanej odpowiedzi: 32,
- akumulacja wyników ewaluacji: 4
- generowanie wyników podczas ewaluacji

W treningu nie używamy optymalizacji FP16! Jej użycie spowoduje, że model nie będzie się trenował. Jeśli chcesz użyć optymalizacji, to możesz skorzystać z **BF16**.

Argumenty powinny również wskazywać, że przeprowadzoany jest proces uczenia i ewaluacji.

In [48]:
```
arguments = Seq2SeqTrainingArguments(
    output_dir=path + "/output-question-answering",
    do_train=True,
    do_eval=True,
    evaluation_strategy="steps",
    eval_steps=100,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=1e-04,
    num_train_epochs=3,
    optim="adafactor",
    logging_first_step=True,
    logging_strategy="steps",
    logging_steps=50,
    save_strategy="epoch",
    generation_max_length=32,
    eval_accumulation_steps=4,
)
```

Zadanie 7 (0.5 punktu)

Utwórz obiekt trenujący `Seq2SeqTrainer`, za pomocą którego będzie trenowany model odpowiadający na pytania.

Obiekt ten powinien:

- wykorzystywać model `plt5-base`,
- wykorzystywać zbiór `train` do treningu,
- wykorzystawać zbiór `dev` do evaluacji,
- wykorzystać klasę batchującą (`data_collator`) o nazwie `DataCollatorWithPadding`.

In [49]:
```
from transformers import DataCollatorWithPadding

trainer = Seq2SeqTrainer(
    model=model,
    args=arguments,
    train_dataset=tokenized_datasets["train"].shuffle(seed=42),
    eval_dataset=tokenized_datasets["dev"].shuffle(seed=42),
    compute_metrics=compute_metrics,
    data_collator=DataCollatorWithPadding(tokenizer=plt5_tokenizer)
)
```

In [50]:
```
%load_ext tensorboard
%tensorboard --logdir gdrive/MyDrive/poquad/output-question-answering/run
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

Mając przygotowane wszystkie dane wejściowe możemy rozpocząć proces treningu.

Uwaga: proces treningu na Google Colab z wykorzystaniem akceleratora zajmuje ok. 3 godziny. Uruchomienie treningu na CPU może trwać ponad 1 dzień!

Możesz pominąć ten proces i w kolejnych krokach wykorzystać gotowy model `apohllo/plt5-base-poquad`, który znajduje się w repozytorium Hugginface.

In [51]:
```
# trainer.train()
```

Zadanie 8 (1.5 punkt)

Korzystając z wywołania `generate` w modelu, wygeneruj odpowiedzi dla 1 kontekstu i 10 pytań dotyczących tego kontekstu. Pamiętaj aby zamienić identyfikatory tokenów na ich treść. Możesz do tygo wykorzystać wywołanie `decode` z tokenizera.

Jeśli w poprzednim punkcie nie udało Ci się wytrenować modelu, możesz skorzystać z modelu `apohllo/plt5-base-poquad`.

Oceń wyniki (odpowiedzi) generowane przez model.

In [52]:
```
from transformers import AutoModelForSeq2SeqLM

model_apohllo = AutoModelForSeq2SeqLM.from_pretrained("apohllo/plt5-base-
```

```
config.json:   0%|          | 0.00/826 [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/1.10G [00:00<?, ?B/s]
generation_config.json:   0%|          | 0.00/112 [00:00<?, ?B/s]
```

In [53]:
```
context = """
Ratel miodożerny, ratel, miodożer, daw. pszczoło-jamnik (Mellivora capens
– gatunek drapieżnego ssaka z rodziny łasicowatych, zamieszkujący lasy,
zarośla, stepy i sawanny Afryki na zachód, wschód i południe od Sahary,
oprócz dżungli Afryki Środkowej, Azję Południową, aż po Nepal. Jest jedyn
przedstawicielem rodzaju Mellivora.

Osiąga długość ciała 60–70 cm, ogona 20–30 cm. Wyglądem, wielkością
```

```
i sposobem poruszania się najbardziej przypomina rosomaka tundrowego. Moc
budowa ciała, krótkie i muskularne kończyny, krótki ogon, sierść długa,
sztywna, ubarwienie grzbietu srebrzystobiałe, spód i boki ciała
brunatnoczarne, na granicy obydwu barw biała pręga, w przednich łapach dł
pazury.

Skóra na grzbiecie gruba i sztywna stanowi swoistą tarczę przed rozmaitym
ukąszeniami, np. węży, innych drapieżników, przed ukłuciami pszczół, jest
ona niezwykle luźna dzięki rozwiniętej podskórnej tkance tłuszczowej,
co stanowi dodatkowy atut obronny.
"""

questions = [
    "Wymień inne nazwy na ratela miodożernego.",
    "Jak dawniej nazywał się ratel?",
    "Czy miodożer jest ssakiem?",
    "Do jakiej rodziny należy miodożer?",
    "Czy ratel miodożerny jest drapieżnikiem?",
    "Jaki jest obszar występowania pszczoło-jamnika?",
    "Zdefiniuj charakterystykę fizyczną ratela (budowa ciała).",
    "Czy ratel ma pazury?",
    "Co jest główną bronią defensywną miodożera? Dlaczego?",
    "Przed czym między innymi chroni gruba skóra ratela miodożernego?",
]

for q in questions:
  question_encoded = plt5_tokenizer.encode(
    f"Pytanie: {q} Kontekst: {context}",
    return_tensors="pt",
    truncation=True,
    max_length=256,
  )
  answer_encoded = model_apohllo.generate(question_encoded)[0]
  answer_encoded = answer_encoded[1:answer_encoded.shape[0] - 1]
  answer_decoded = plt5_tokenizer.decode(answer_encoded)

  print(f"Pytanie: {q}")
  print(f"Odpowiedź: {answer_decoded}")
  print()
```

```
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1
273: UserWarning: Using the model-agnostic default `max_length` (=20) to c
ontrol the generation length. We recommend setting `max_new_tokens` to con
trol the maximum length of the generation.
  warnings.warn(
```

```
Pytanie: Wymień inne nazwy na ratela miodożernego.
Odpowiedź: ratel, miodożer

Pytanie: Jak dawniej nazywał się ratel?
Odpowiedź: miodożer

Pytanie: Czy miodożer jest ssakiem?
Odpowiedź: tak

Pytanie: Do jakiej rodziny należy miodożer?
Odpowiedź: łasicowatych

Pytanie: Czy ratel miodożerny jest drapieżnikiem?
Odpowiedź: tak

Pytanie: Jaki jest obszar występowania pszczoło-jamnika?
Odpowiedź: lasy, zarośla, stepy i sawanny Afryki na zachód, wschód i połud
nie od

Pytanie: Zdefiniuj charakterystykę fizyczną ratela (budowa ciała).
Odpowiedź: Mocna budowa ciała, krótkie i muskularne kończyny, krótki ogon,

Pytanie: Czy ratel ma pazury?
Odpowiedź: tak

Pytanie: Co jest główną bronią defensywną miodożera? Dlaczego?
Odpowiedź: skóra na grzbiecie gruba i sztywna

Pytanie: Przed czym między innymi chroni gruba skóra ratela miodożernego?
Odpowiedź: rozmaitymi ukąszeniami, np. węży, innych drapieżników, przed
```

Komentarz

Model dobrze generuje odpowiedzi, odpowiedzi zamknięte są generowane dokładnie, natomiast odpowiedzi, gdzie potrzeba coś więcej dodać na podstawie kontekstu nie są idealne. Głównym problem jest niedokładność odpowiedzi oraz poprawność gramatyczna.

Zadanie dodatkowe (2 punkty)

Stworzenie pełnego rozwiązania w zakresie odpowiadania na pytania wymaga również znajdowania kontekstów, w których może pojawić się pytanie.

Obenie istnieje coraz więcej modeli neuronalnych, które bardzo dobrze radzą sobie ze znajdowaniem odpowiednich tekstów. Również dla języka polskiego następuje tutaj istotny postęp. Powstała m.in. strona śledząca postępy w tym zakresie.

Korzystając z informacji na tej stronie wybierz jeden z modeli do wyszukiwania kontekstów (najlepiej o rozmiarze `base` lub `small`). Zamień konteksty występujące w zbiorze PoQuAD na reprezentacje wektorowe. To samo zrób z pytaniami występującymi w tym zbiorze. Dla każdego pytania znajdź kontekst, który według modelu najlepiej odpowiada na zadane pytanie. Do znalezienia kontekstu oblicz iloczyn skalarny pomiędzy reprezentacją pytania oraz wszystkimi kontekstami

ze zbioru. Następnie uruchom model generujący odpowiedź na znalezionym kontekście. Porównaj wyniki uzyskiwane w ten sposób, z wynikami, gdy poprawny kontekst jest znany.

W celu przyspieszenie obliczeń możesz zmniejszyć liczbę pytań i odpowiadających im kontekstów. Pamiętaj jednak, żeby liczba kontekstów była odpowiednio duża (sugerowana wartość min. to 1000 kontekstów), tak żeby znalezienie kontekstu nie było trywialne.