import torch

import numpy as np

from transformers import AutoTokenizer, AutoModelForQuestionAnswering

tokenizer = AutoTokenizer.from\_pretrained("timpal0l/mdeberta-v3-base-squad2")

model = AutoModelForQuestionAnswering.from\_pretrained("timpal0l/mdeberta-v3-base-squad2")

question = 'Кто приехал в гости к Анне Павловне?'

text ='''

 Вся высшая знать Петербурга прибыла в гости к Анне Павловне.

'''

tokenized = tokenizer.encode\_plus(

   question, text,

   add\_special\_tokens=False

)

tokens = tokenizer.convert\_ids\_to\_tokens(tokenized['input\_ids'])

# Общая длина каждого блока

max\_chunk\_length = 512

# Длина наложения

overlapped\_length = 30

# Длина вопроса в токенах

answer\_tokens\_length = tokenized.token\_type\_ids.count(0)

# Токены вопроса, закодированные числами

answer\_input\_ids = tokenized.input\_ids[:answer\_tokens\_length]

# Длина основного текста первого блока без наложения

first\_context\_chunk\_length = max\_chunk\_length - answer\_tokens\_length

# Длина основного текста остальных блоков с наложением

context\_chunk\_length = max\_chunk\_length - answer\_tokens\_length - overlapped\_length

# Токены основного текста

context\_input\_ids = tokenized.input\_ids[answer\_tokens\_length:]

# Основной текст первого блока

first = context\_input\_ids[:first\_context\_chunk\_length]

# Основной текст остальных блоков

others = context\_input\_ids[first\_context\_chunk\_length:]

# Если есть блоки кроме первого

# тогда обрабатываются все блоки

if len(others) > 0:

  # Кол-во нулевых токенов, для выравнивания последнего блока по длине

  padding\_length = context\_chunk\_length - (len(others) % context\_chunk\_length)

  others += [0] \* padding\_length

  # Кол-во блоков и их длина без добавления наложения

  new\_size = (

      len(others) // context\_chunk\_length,

      context\_chunk\_length

  )

  # Упаковка блоков

  new\_context\_input\_ids = np.reshape(others, new\_size)

  # Вычисление наложения

  overlappeds = new\_context\_input\_ids[:, -overlapped\_length:]

  # Добавление в наложения частей из первого блока

  overlappeds = np.insert(overlappeds, 0, first[-overlapped\_length:], axis=0)

  # Удаление наложение из последнего блока, так как оно не нужно

  overlappeds = overlappeds[:-1]

  # Добавление наложения

  new\_context\_input\_ids = np.c\_[overlappeds, new\_context\_input\_ids]

  # Добавление первого блока

  new\_context\_input\_ids = np.insert(new\_context\_input\_ids, 0, first, axis=0)

  # Добавление вопроса в каждый блок

  new\_input\_ids = np.c\_[

    [answer\_input\_ids] \* new\_context\_input\_ids.shape[0],

    new\_context\_input\_ids

  ]

# иначе обрабатывается только первый

else:

  # Кол-во нулевых токенов, для выравнивания блока по длине

  padding\_length = first\_context\_chunk\_length - (len(first) % first\_context\_chunk\_length)

  # Добавление нулевых токенов

  new\_input\_ids = np.array(

    [answer\_input\_ids + first + [0] \* padding\_length]

  )

  # Кол-во блоков

count\_chunks = new\_input\_ids.shape[0]

# Маска, разделяющая вопрос и текст

new\_token\_type\_ids = [

  # вопрос блока

  [0] \* answer\_tokens\_length

  # текст блока

  + [1] \* (max\_chunk\_length - answer\_tokens\_length)

] \* count\_chunks

# Маска "внимания" модели на все токены, кроме нулевых в последнем блоке

new\_attention\_mask = (

  # во всех блоках, кроме последнего, "внимание" на все слова

  [[1] \* max\_chunk\_length] \* (count\_chunks - 1)

  # в последнем блоке "внимание" только на ненулевые токены

  + [([1] \* (max\_chunk\_length - padding\_length)) + ([0] \* padding\_length)]

)

# Токенизированный текст в виде блоков, упакованный в torch

new\_tokenized = {

 'input\_ids': torch.tensor(new\_input\_ids),

 'token\_type\_ids': torch.tensor(new\_token\_type\_ids),

 'attention\_mask': torch.tensor(new\_attention\_mask)

}

outputs = model(\*\*new\_tokenized)

# Позиции в 2D списке токенов начала и конца наиболее вероятного ответа

# позиции одним числом

start\_index = torch.argmax(outputs.start\_logits)

end\_index = torch.argmax(outputs.end\_logits)

# Пересчёт позиций начала и конца ответа для 1D списка токенов

# = длина первого блока + (

#   позиция - длина первого блока

#   - длина ответов и отступов во всех блоках, кроме первого

# )

start\_index = max\_chunk\_length + (

  start\_index - max\_chunk\_length

  - (answer\_tokens\_length + overlapped\_length)

  \* (start\_index // max\_chunk\_length)

)

end\_index = max\_chunk\_length + (

  end\_index - max\_chunk\_length

  - (answer\_tokens\_length + overlapped\_length)

  \* (end\_index // max\_chunk\_length)

)

# Составление ответа

# если есть символ начала слова '▁', то он заменяется на пробел

#

answer = ''.join([t.replace('▁', ' ') for t in tokens[start\_index:end\_index+1]])

#answer = ''.join([t.replace('\_', ' ') for t in tokens[start\_index:end\_index+1]])

print('Вопрос:', question)

print('Ответ:', answer)