Импортируем необходимые библиотеки:

!pip install fastapi

!pip install transformers

!pip install pydantic

!pip install TensorFlow

!pip install Torch

!pip install sentencepiece

import os  # Импорт модуля для работы с операционной системой

import nltk  # Импорт библиотеки Natural Language Toolkit (NLTK)

import matplotlib.pyplot as plt  # Импорт библиотеки для построения графиков

import csv  # Импорт модуля для работы с файлами CSV

from nltk.corpus import stopwords  # Импорт списка стоп-слов из NLTK

from nltk.stem import SnowballStemmer  # Импорт стеммера из NLTK

from transformers import pipeline  # Импорт модуля pipeline из библиотеки transformers

from nltk.tokenize import word\_tokenize  # Функция для токенизации текста

from nltk.translate.bleu\_score import corpus\_bleu  # Функция для вычисления BLEU

from nltk.translate.meteor\_score import meteor\_score  # Функция для вычисления METEOR

from rouge import Rouge  # Класс для вычисления метрики ROUGE

# Загрузка ресурсов NLTK

nltk.download('stopwords')  # Загрузка списка стоп-слов из NLTK

nltk.download('punkt')  # Загрузка пунктуации из NLTK

# Загрузка модели для перевода текста с английского на русский

translator = pipeline("translation\_en\_to\_ru", "Helsinki-NLP/opus-mt-en-ru")

Эта функция разделяет текст из исходного файла на несколько файлов, где каждый файл содержит не более заданного количества токенов (по умолчанию 50). Текст сначала токенизируется на предложения, затем предложения разбиваются на файлы таким образом, чтобы каждый файл содержал не более максимального числа токенов. Если в тексте остались предложения после разделения на файлы, они записываются в последний файл:

def split\_text\_to\_files(input\_file\_path, output\_directory, max\_tokens\_per\_file=50):

    # Читаем исходный текстовый файл

    with open(input\_file\_path, 'r', encoding='utf-8') as f:

        text = f.read()

    # Создаем директорию для сохранения разделенных текстовых файлов

    if not os.path.exists(output\_directory):

        os.makedirs(output\_directory)

    # Токенизируем текст на предложения

    sentences = nltk.sent\_tokenize(text)

    # Разбиваем предложения на файлы по max\_tokens\_per\_file токенов

    file\_count = 1

    tokens\_count = 0

    current\_sentences = []

    for sentence in sentences:

        sentence\_tokens = nltk.word\_tokenize(sentence)

        # Добавляем предложение в список перед проверкой токенов

        current\_sentences.append(sentence)

        if tokens\_count + len(sentence\_tokens) <= max\_tokens\_per\_file:

            tokens\_count += len(sentence\_tokens)

        else:

            # Создаем путь для нового файла

            output\_file\_path = os.path.join(output\_directory, f"input\_text ({file\_count}).txt")

            # Записываем предложения в новый файл

            with open(output\_file\_path, 'w', encoding='utf-8') as f:

                f.write('\n'.join(current\_sentences))

            # Сбрасываем счетчики для следующего файла

            file\_count += 1

            current\_sentences = [sentence]

            tokens\_count = len(sentence\_tokens)

    # Записываем оставшиеся предложения в последний файл

    if current\_sentences:

        output\_file\_path = os.path.join(output\_directory, f"input\_text ({file\_count}).txt")

        with open(output\_file\_path, 'w', encoding='utf-8') as f:

            f.write('\n'.join(current\_sentences))

# Пример использования:

input\_file\_path = "input\_text.txt"

output\_directory = "input\_texts"

split\_text\_to\_files(input\_file\_path, output\_directory)

Этот код выполняет `предобработку текста`, включая токенизацию, удаление стоп-слов на русском языке, стемминг токенов и ограничение количества токенов до 512, а затем возвращает предобработанный текст:

# Создание экземпляра стеммера

stemmer = SnowballStemmer("russian")

# Получение списка стоп-слов на русском языке

stop\_words = set(stopwords.words('russian'))

# Функция предобработки текста

def preprocess\_text(text):

    # Токенизация текста

    tokens = word\_tokenize(text)

    # Удаление стоп-слов

    filtered\_tokens = [word for word in tokens if word.lower() not in stop\_words]

    # Стемминг токенов

    stemmed\_tokens = [stemmer.stem(token) for token in filtered\_tokens]

    # Ограничение количества токенов до 512

    limited\_tokens = stemmed\_tokens[:512]

    # Склеивание токенов обратно в строку

    preprocessed\_text = ' '.join(limited\_tokens)

    return preprocessed\_text

Этот код выполняет `перевод текста из файла с английского на русский`, `предобрабатывает его`, используя функцию `preprocess\_text`, затем `сохраняет результат перевода в файл`. После этого определяются пути к директориям с входными текстовыми файлами и для сохранения переведенных текстовых файлов:

# Функция перевода текста из файла и сохранения перевода в файл

def translate\_file(input\_file, output\_file):

    """Translate English text from input file to Russian, preprocess it, and save to output file"""

    with open(input\_file, 'r', encoding='utf-8') as f:

        input\_text = f.read()  # Чтение текста из входного файла

    # Предобработка текста

    preprocessed\_text = preprocess\_text(input\_text)

    # Перевод предобработанного текста

    translated\_text = translator(preprocessed\_text)[0]['translation\_text']

    with open(output\_file, 'w', encoding='utf-8') as f:

        f.write(translated\_text)  # Запись переведенного текста в файл

# Получаем путь к текущей директории (директории, в которой находится блокнот)

current\_directory = os.getcwd()

# Путь к директории с входными текстовыми файлами (предполагая, что они находятся в поддиректории "input\_texts")

input\_directory = os.path.join(current\_directory, "input\_texts")

# Путь к директории для сохранения переведенных текстовых файлов (предполагая, что они будут сохранены в поддиректории "translated\_texts")

output\_directory = os.path.join(current\_directory, "translated\_texts")

Этот код создает директорию для сохранения `переведенных файлов`, если таковая еще не существует, затем получает список всех файлов в директории с исходными текстами.

Затем он `переводит каждый файл` и сохраняет результат в директорию с переведенными текстами, преобразуя названия файлов.

После этого он получает список всех файлов в директории с переведенными текстами и `создает общий файл`, в который записываются все переведенные тексты.

# Создаем директорию для сохранения переведенных текстовых файлов, если она еще не существует

if not os.path.exists(output\_directory):

    os.makedirs(output\_directory)

# Получаем список всех файлов в директории с входными текстами

input\_files = os.listdir(input\_directory)

# Переводим каждый файл и сохраняем результат в директорию с выходными текстами

for input\_file in input\_files:

    # Формируем пути к входному и выходному файлам

    input\_file\_path = os.path.join(input\_directory, input\_file)

    output\_file\_path = os.path.join(output\_directory, input\_file.replace("input\_text", "output\_text"))

    # Переводим текст из входного файла и сохраняем результат в выходной файл

    translate\_file(input\_file\_path, output\_file\_path)

# Получаем список всех файлов в директории с переведенными текстами

translated\_files = os.listdir(output\_directory)

# Путь к общему файлу, куда будут записаны все переведенные тексты

combined\_output\_file = os.path.join(current\_directory, "translated.txt")

Этот блок кода открывает `общий файл для записи` и проходит по каждому переведенному файлу. Для каждого файла он читает его содержимое и записывает его в общий файл, добавляя новую строку после каждого перевода.

# Открываем общий файл для записи

with open(combined\_output\_file, 'w', encoding='utf-8') as combined\_file:

    # Проходим по каждому переведенному файлу

    for translated\_file in translated\_files:

        # Формируем путь к текущему переведенному файлу

        translated\_file\_path = os.path.join(output\_directory, translated\_file)

        # Читаем содержимое текущего переведенного файла

        with open(translated\_file\_path, 'r', encoding='utf-8') as f:

            translated\_text = f.read()

        # Записываем содержимое текущего переведенного файла в общий файл

        combined\_file.write(translated\_text + '\n')

Этот код определяет функцию `load\_text\_preprocessed`, которая `загружает текст из файла и предварительно обрабатывает его`. Затем он `загружает эталонный и переведенный тексты`, предварительно обрабатывая их с помощью этой функции. Далее происходит токенизация обоих текстовых файлов с использованием нижнего регистра.

# Функция для загрузки текста из файла с предобработкой

def load\_text\_preprocessed(file\_path):

    with open(file\_path, 'r', encoding='utf-8') as file:

        text = file.read()

        # Предобработка загруженного текста

        preprocessed\_text = preprocess\_text(text)

    return preprocessed\_text

# Загрузка предобработанного эталонного текста

reference\_text = load\_text\_preprocessed('reference.txt')

# Загрузка предобработанного переведенного текста

translated\_text = load\_text\_preprocessed('translated.txt')

# Токенизация текстов

reference\_tokens = word\_tokenize(reference\_text.lower())

translated\_tokens = word\_tokenize(translated\_text.lower())

Этот фрагмент кода вычисляет `BLEU-метрику` для оценки качества перевода, используя эталонные и переведенные токены. Затем он создает экземпляр класса Rouge для вычисления метрики `ROUGE` и вычисляет ROUGE-оценки для переведенного и эталонного текстов. Наконец, он использует метрику `METEOR` для оценки качества перевода и выводит результаты всех трех метрик.

1. BLEU-метрика: оценивает качество перевода путем сравнения переведенного текста с эталонным текстом на основе n-грамм.

2. ROUGE-метрика: вычисляет сходство между двумя текстами, учитывая пересечение слов и последовательность слов.

3. METEOR-метрика: оценивает качество перевода, учитывая семантическое сходство между переведенным и эталонным текстами.

# Вычисление BLEU-метрики

bleu\_score = corpus\_bleu([[reference\_tokens]], [translated\_tokens])

# Создание экземпляра класса Rouge

rouge = Rouge()

# Вычисление метрики ROUGE

scores = rouge.get\_scores(translated\_text, reference\_text)

# Оценка качества перевода с помощью метрики METEOR

score = meteor\_score([reference\_tokens], translated\_tokens)

# Вывод результатов

print("BLEU score:", bleu\_score)

print("ROUGE scores:", scores)

print("METEOR score:", score)

Этот блок кода вычисляет `среднее арифметическое из всех метрик`, включая `BLEU, ROUGE-1, ROUGE-2, ROUGE-L и METEOR`, и выводит результат. Для ROUGE-метрик происходит суммирование значений F-меры для каждой метрики из всех результатов ROUGE, а затем деление на общее количество метрик.

# Вычисление среднего арифметического из всех метрик

average = (bleu\_score + sum(rouge['rouge-1']['f'] for rouge in scores) + sum(rouge['rouge-2']['f'] for rouge in scores) + sum(rouge['rouge-l']['f'] for rouge in scores) + score) / 5

# Вывод среднего арифметического из всех метрик

print("Среднее арифметическое из всех метрик:", average)

Этот код вычисляет сумму всех метрик, включая `BLEU, ROUGE-1 и METEOR`, и умножает каждую метрику на соответствующий ей `уникальный вес`, затем суммирует результаты для получения метрики оценки качества перевода художественного текста. Результат выводится как оценка качества перевода `художественного текста`:

# Вычисление суммы всех метрик

k = bleu\_score + sum(rouge['rouge-1']['f'] for rouge in scores) + score

# Уникальные веса для каждой метрики оценки качества перевода художественных текстов

w\_bleu = 0.1

w\_rouge\_1 = 0.5

w\_score = 0.4

# Вычисление метрики оценки качества перевода художественных текстов с уникальными весами

art\_met = w\_bleu \* bleu\_score + w\_rouge\_1 \* sum(rouge['rouge-1']['f'] for rouge in scores) + w\_score \* score

# Результаты оценки качества перевода художественного текста

print("Оценка качества перевода художественного текста:", art\_met)

Этот код определяет функцию `save\_metrics\_to\_csv`, которая сохраняет метрики в CSV-файл. Она проверяет, существует ли файл CSV, и `записывает метрики в CSV-файл`, добавляя заголовки полей, если файл только что создан. Затем он используется для сохранения заданных метрик в файл с указанным именем:

def save\_metrics\_to\_csv(metrics, csv\_file):

    # Проверяем, существует ли файл CSV

    file\_exists = os.path.exists(csv\_file)

    # Открываем файл CSV в режиме добавления данных

    with open(csv\_file, 'a', newline='', encoding='utf-8') as file:

        writer = csv.DictWriter(file, fieldnames=metrics.keys())

        # Если файл не существует, записываем заголовки полей

        if not file\_exists:

            writer.writeheader()

        # Записываем значения метрик в файл CSV

        writer.writerow(metrics)

# Пример использования:

metrics = {

    'BLEU score': bleu\_score,

    'ROUGE scores': scores,

    'METEOR score': score,

    'Среднее арифметическое из всех метрик': average,

    'Оценка качества перевода': art\_met

}

# Путь к файлу CSV

csv\_file = 'En-Ru metrics.csv'

# Сохраняем метрики в файл CSV

save\_metrics\_to\_csv(metrics, csv\_file)

Этот код определяет функцию `load\_metrics\_from\_csv`, которая `загружает метрики из CSV-файла`. Она читает файл CSV и `создает список словарей`, где каждый словарь представляет собой строку в файле CSV (с заголовками столбцов в качестве ключей). Функция возвращает этот список словарей, представляющих метрики из файла CSV.

Пример использования этой функции загружает метрики из файла 'En-Ru metrics.csv' и выводит их на экран:

def load\_metrics\_from\_csv(csv\_file):

    metrics = []

    with open(csv\_file, 'r', newline='', encoding='utf-8') as file:

        reader = csv.DictReader(file)

        for row in reader:

            metrics.append(row)

    return metrics

# Пример использования:

csv\_file = 'En-Ru metrics.csv'

# Загружаем данные из файла CSV

metrics = load\_metrics\_from\_csv(csv\_file)

# Отображаем значения метрик

for entry in metrics:

    print(entry)

Этот код `загружает метрики` из CSV-файла, `вычисляет среднее арифметическое` для BLEU score, METEOR score, ROUGE-1, ROUGE-2, ROUGE-L и `метрики с уникальными весами`, а затем выводит результаты на экран:

def load\_metrics\_from\_csv(csv\_file):

    metrics = []

    with open(csv\_file, 'r', newline='', encoding='utf-8') as file:

        reader = csv.DictReader(file)

        for row in reader:

            metrics.append(row)

    return metrics

def calculate\_average(metrics):

    bleu\_scores = []

    meteor\_scores = []

    rouge\_1\_scores = []

    rouge\_2\_scores = []

    rouge\_l\_scores = []

    average\_scores = []

    art\_met\_scores = []  # Массив для хранения значений art\_met

    for entry in metrics:

        bleu\_scores.append(float(entry['BLEU score']))

        meteor\_scores.append(float(entry['METEOR score']))

        # ROUGE scores в формате строки, преобразуем в словарь

        rouge\_scores = eval(entry['ROUGE scores'])

        rouge\_1\_scores.append(rouge\_scores[0]['rouge-1']['f'])

        rouge\_2\_scores.append(rouge\_scores[0]['rouge-2']['f'])

        rouge\_l\_scores.append(rouge\_scores[0]['rouge-l']['f'])

        average\_scores.append(float(entry['Среднее арифметическое из всех метрик']))

        art\_met\_scores.append(float(entry['Оценка качества перевода']))  # Добавляем значение art\_met из CSV

    # Вычисляем среднее арифметическое для каждой метрики

    average\_bleu = sum(bleu\_scores) / len(bleu\_scores)

    average\_meteor = sum(meteor\_scores) / len(meteor\_scores)

    average\_rouge\_1 = sum(rouge\_1\_scores) / len(rouge\_1\_scores)

    average\_rouge\_2 = sum(rouge\_2\_scores) / len(rouge\_2\_scores)

    average\_rouge\_l = sum(rouge\_l\_scores) / len(rouge\_l\_scores)

    average = sum(average\_scores) / len(average\_scores)

    average\_art\_met = sum(art\_met\_scores) / len(art\_met\_scores)  # Среднее для art\_met

    return average\_bleu, average\_meteor, average\_rouge\_1, average\_rouge\_2, average\_rouge\_l, average, average\_art\_met

# Пример использования:

csv\_file = 'En-Ru metrics.csv'

# Загружаем данные из файла CSV

metrics = load\_metrics\_from\_csv(csv\_file)

# Вычисляем среднее арифметическое

average\_bleu, average\_meteor, average\_rouge\_1, average\_rouge\_2, average\_rouge\_l, average, average\_art\_met = calculate\_average(metrics)

# Выводим среднее арифметическое

print("Среднее арифметическое для BLEU score:", average\_bleu)

print("Среднее арифметическое для METEOR score:", average\_meteor)

print("Среднее арифметическое для ROUGE-1:", average\_rouge\_1)

print("Среднее арифметическое для ROUGE-2:", average\_rouge\_2)

print("Среднее арифметическое для ROUGE-L:", average\_rouge\_l)

print("Среднее арифметическое из всех метрик:", average)

print("Оценка качества перевода:", average\_art\_met)

Этот код создает `гистограмму`, визуализирующую средние значения различных метрик оценки качества перевода, таких как BLEU score, METEOR score и ROUGE scores, а также метрика с весами для группы 2, для наглядного сравнения и анализа:

# Средние значения метрик

metrics\_names = ['BLEU score', 'METEOR score', 'ROUGE-1', 'ROUGE-2', 'ROUGE-L', 'Среднее', 'Оценка качества']

metrics\_values = [average\_bleu, average\_meteor, average\_rouge\_1, average\_rouge\_2, average\_rouge\_l, average, art\_met]

# Создание диаграммы

plt.figure(figsize=(10, 6))

plt.bar(metrics\_names, metrics\_values, color='skyblue')

plt.title('Средние значения метрик')

plt.xlabel('Метрика оценки качества перевода художественного текста')

plt.ylabel('Значение')

plt.xticks(rotation=45, ha='right')  # Поворот подписей по оси X для лучшей читаемости

plt.tight\_layout()

# Отображение диаграммы

plt.show()

Этот фрагмент кода определяет функцию `load\_text`, которая загружает текст из файла и возвращает его содержимое. Затем он загружает эталонный текст из файла с помощью этой функции и выводит результат его предобработки:

# Функция для загрузки текста из файла

def load\_text(file\_path):

    with open(file\_path, 'r', encoding='utf-8') as file:  # Открытие файла для чтения с указанием кодировки UTF-8

        text = file.read()  # Чтение содержимого файла и сохранение в переменную text

    return text  # Возвращение загруженного текста

# Загрузка эталонного текста

reference\_text = load\_text('reference.txt')

# Проверка содержимого файла после предобработки

print("Текст после предобработки:\n", preprocess\_text(reference\_text))

Этот фрагмент кода `загружает и выводит эталонный текст до и после его предобработки`. Затем он загружает и выводит переведенный текст до и после его предобработки:

# Загрузка эталонного текста до и после предобработки

print("Original reference text:")

print(reference\_text)

reference\_text\_preprocessed = preprocess\_text(reference\_text)

print("Preprocessed reference text:")

print(reference\_text\_preprocessed)

# Загрузка переведенного текста до и после предобработки

print("Original translated text:")

print(translated\_text)

translated\_text\_preprocessed = preprocess\_text(translated\_text)

print("Preprocessed translated text:")

print(translated\_text\_preprocessed)