**ОТЧЁТ ПО ПрИнж**

**Ларин Антон Сергеевич  
Каверин Тихон Витальевич**

**Группа 11-210**

**Глава 1 Генерация Текста**

**Описание кода**

Код предназначен для генерации текста с использованием предобученной языковой модели GPT-2 из библиотеки transformers. Модель берёт текстовую подсказку (prompt) и генерирует продолжение текста на её основе.

### ****Подробный анализ****

1. **Импорт библиотек**
   * GPT2LMHeadModel и GPT2Tokenizer импортируются из библиотеки transformers.  
     Эти классы используются для работы с моделью GPT-2 и её токенайзером.
2. **Инициализация модели и токенайзера**
   * Используется модель GPT-2 (model\_name = "gpt2"), которая загружается из предобученного хранилища с помощью метода from\_pretrained().
   * Токенайзер также инициализируется с помощью from\_pretrained().
3. **Установка pad\_token**
   * Так как GPT-2 изначально не поддерживает pad\_token, его необходимо задать вручную, чтобы обеспечить корректную обработку текста. В данном случае используется токен конца строки (eos\_token) в качестве заполнителя.
4. **Подготовка текстового ввода**
   * Подсказка "Science and technology of the future" преобразуется в тензор идентификаторов токенов с помощью tokenizer.encode().
   * Создаётся attention\_mask, которая маскирует пустые токены (pad\_token).
5. **Генерация текста**  
   Метод model.generate() используется для генерации продолжения текста с заданными параметрами:
   * max\_length=500: Ограничение на длину сгенерированного текста.
   * temperature=0.7: Параметр температуры для регулировки случайности при выборе следующего токена.
   * top\_p=0.9: Используется метод nucleus sampling для выбора токенов, чья совокупная вероятность превышает 90%.
   * do\_sample=True: Включение случайного семплирования токенов.
   * pad\_token\_id: Указание идентификатора заполнителя для корректной генерации.
6. **Декодирование текста**
   * Результат генерации (в формате токенов) декодируется обратно в текст с помощью tokenizer.decode().
   * Специальные токены (<pad>, <eos>) исключаются из декодированного текста (skip\_special\_tokens=True).
7. **Вывод результата**
   * Сгенерированный текст выводится в консоль.

**Ключевые особенности**

* **Универсальность**: Код работает с любой предобученной моделью GPT-2 благодаря использованию метода from\_pretrained().
* **Настраиваемость**: Гибкие параметры для генерации текста (например, длина, температура, топ-P).
* **Совместимость**: Учитываются ограничения модели GPT-2, такие как отсутствие pad\_token по умолчанию.

**Глава 2 Анализ Изображения**

#### **1. Импорт библиотек**

* **ViTForImageClassification и ViTImageProcessor**: Используются для загрузки предобученной модели Vision Transformer (ViT) и обработки изображений.
* **PIL.Image**: Библиотека для работы с изображениями.
* **torch**: Для представления данных в виде тензоров, необходимых для обработки в модели.

#### **2. Загрузка модели и процессора**

model = ViTForImageClassification.from\_pretrained("google/vit-base-patch16-224")

processor = ViTImageProcessor.from\_pretrained("google/vit-base-patch16-224")

* **Модель**: Загружается предобученная ViT-модель (vit-base-patch16-224), которая классифицирует изображения.
* **Процессор**: Используется для предварительной обработки изображений (масштабирование, нормализация и преобразование в тензор).

#### **3. Обработка изображения**

image = Image.open("image4.jpg")

inputs = processor(images=image, return\_tensors="pt")

* **Image.open()**: Открывает изображение из указанного файла.
* **processor()**: Преобразует изображение в формат, подходящий для модели:
  + Масштабирование до нужного размера (224x224 пикселя).
  + Нормализация пикселей.
  + Преобразование в тензор PyTorch (return\_tensors="pt").

#### **4. Предсказание класса**

outputs = model(\*\*inputs)

predicted\_class\_idx = outputs.logits.argmax(-1).item()

* **model(\*\*inputs)**: Передаёт обработанные данные в модель для классификации.
* **outputs.logits**: Возвращает логиты — необработанные предсказания модели для каждого класса.
* **argmax(-1)**: Находит индекс класса с наивысшей вероятностью.

#### **5. Получение названия класса**

predicted\_class\_label = model.config.id2label[predicted\_class\_idx]

* **id2label**: Словарь, который сопоставляет индексы классов с их текстовыми названиями (например, "dog", "cat").
* Полученное название класса выводится в консоль.

### ****Ключевые особенности****

1. **Мощная архитектура ViT**  
   Vision Transformer (ViT) — это современная архитектура для обработки изображений, вдохновлённая механизмами внимания в NLP.
2. **Простая интеграция**  
   Библиотека transformers предоставляет высокоуровневые инструменты для работы с предобученными моделями, что упрощает процесс.
3. **Автоматическая обработка данных**  
   Использование ViTImageProcessor избавляет от необходимости вручную нормализовать изображения и приводить их к нужному размеру.

**Глава 3 Звук в текст**

### ****Описание****

Код выполняет задачу распознавания речи в реальном времени, записывая аудио с микрофона и преобразуя его в текст с помощью Google Web Speech API через библиотеку speech\_recognition.

### ****Анализ работы****

#### **1. Импорт библиотеки**

* **speech\_recognition** (sr): популярная библиотека Python для распознавания речи, поддерживающая различные API, включая Google Web Speech API.

#### **2. Инициализация распознавателя и микрофона**

recognizer = sr.Recognizer()

microphone = sr.Microphone()

* **Recognizer**: основной объект для управления процессом распознавания.
* **Microphone**: объект для взаимодействия с микрофоном устройства.

#### **3. Функция для распознавания речи**

python

Копировать код

def recognize\_speech\_from\_microphone():

* Основной функционал для записи и преобразования речи в текст.

#### **4. Пошаговая логика функции**

1. **Настройка микрофона**

recognizer.adjust\_for\_ambient\_noise(source)

* + Устанавливает порог шума для фильтрации фоновых звуков.
  + Рекомендуется говорить в тишине для повышения точности.

1. **Запись аудио**

audio\_data = recognizer.listen(source)

* + Захватывает аудио с микрофона и сохраняет в формате AudioData.

1. **Распознавание речи**

text = recognizer.recognize\_google(audio\_data, language="ru-RU")

* + Использует Google Web Speech API для преобразования аудио в текст.
  + Указан язык распознавания: **русский** (ru-RU).

1. **Обработка ошибок**
   * **UnknownValueError**: Если речь не может быть распознана.
   * **RequestError**: Если произошла ошибка соединения с API.

#### **5. Запуск функции**

recognize\_speech\_from\_microphone()

* Функция вызывается, когда код исполняется.

### ****Ключевые особенности****

1. **Лёгкость использования**  
   Библиотека speech\_recognition обеспечивает высокоуровневый интерфейс для распознавания речи.
2. **Реализация с микрофоном**  
   Код взаимодействует с микрофоном устройства, автоматически калибрует его и записывает звук.
3. **Google Web Speech API**  
   Используется облачная служба для точного распознавания речи.

**Глава 4 Трекинг Видео**

### ****Описание****

Этот код выполняет задачу детекции объектов на каждом кадре видео. Он использует модель **DETR** (DEtection TRansformer) для обнаружения объектов на изображениях. Результаты отображаются визуально, а обработанное видео сохраняется в файл.

### ****Анализ работы****

#### **1. Импорт библиотек**

import cv2

from transformers import DetrImageProcessor, DetrForObjectDetection

from PIL import Image

import torch

* **cv2**: используется для работы с видео и изображениями (загрузка, обработка, запись).
* **transformers**: библиотека для работы с моделью DETR.
* **PIL.Image**: для конвертации изображений между форматами.
* **torch**: для работы с тензорами (входы/выходы модели).

#### **2. Загрузка модели и процессора**

processor = DetrImageProcessor.from\_pretrained("facebook/detr-resnet-50")

model = DetrForObjectDetection.from\_pretrained("facebook/detr-resnet-50")

* **DetrImageProcessor**: преобразует изображения в формат, подходящий для модели DETR.
* **DetrForObjectDetection**: модель, обученная на обнаружение объектов.

#### **3. Функция для обработки одного кадра**

def detect\_objects\_on\_frame(frame):

Функция выполняет следующие действия:

1. Конвертация кадра из формата **OpenCV** в формат **PIL.Image**.

image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

1. Подготовка входных данных для модели:

inputs = processor(images=image, return\_tensors="pt")

1. Постпроцессинг результатов для получения координат объектов, меток и вероятностей:

results = processor.post\_process\_object\_detection(outputs, target\_sizes=target\_sizes, threshold=0.9)

1. Рисование рамок и меток на кадре:

cv2.rectangle(...) # Рамки

cv2.putText(...) # Метки с вероятностями

#### **4. Работа с видео**

1. **Чтение видеофайла**:

cap = cv2.VideoCapture(video\_path)

* + Проверяется, открылся ли файл корректно.

1. **Настройка параметров для записи**:

out = cv2.VideoWriter(output\_path, fourcc, fps, (frame\_width, frame\_height))

* + Кодек: **mp4v**.
  + Частота кадров: определяется из исходного видео.
  + Размеры кадра: совпадают с исходным видео.

1. **Обработка кадров**:

while cap.isOpened():

ret, frame = cap.read()

* + Цикл считывает кадры из видео и передаёт их в функцию detect\_objects\_on\_frame.
  + Обработанные кадры записываются в файл.

1. **Отображение результатов**:

cv2.imshow("Detection", frame\_with\_objects)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

* + Обработанные кадры можно наблюдать в реальном времени. Нажатие Q завершает процесс.

1. **Завершение работы**:

cap.release()

out.release()

cv2.destroyAllWindows()

* + Закрытие всех ресурсов и окон.

### ****Ключевые особенности****

1. **Использование DETR**: мощная модель, объединяющая CNN и Transformer, для точного обнаружения объектов.
2. **Обработка видео**: весь процесс от кадра до готового видео.
3. **Интерактивность**: возможность наблюдать результаты в реальном времени.