Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

УДК 004.94

Работа защищена
«___» _____ 2025 г.
Оценка _____
Председатель ГЭК, д.т.н., проф.
_____ С. П. Пронин

Допустить к защите
«___» _____ 2025 г.
Заведующий кафедрой ВТиЭ,
к.ф.-м.н., доцент
_____ В. В. Пашнев

СЕРВИС АВТОМАТИЗИРОВАННОГО РЕШЕНИЯ САРТСНА

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

БР 09.03.01.5.306М.1337 ПЗ

Студент группы 5.306М А. В. Лаптев

Руководитель работы к.ф.-м.н., доцент В. В. Электроник

Консультанты:

Нормоконтролер к.ф.-м.н., доцент А. В. Калачёв

Барнаул 2025

РЕФЕРАТ

Объем работы листов 22

Количество рисунков 6

Количество используемых источников 5

Количество таблиц 6

КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ, СИСТЕМА УПРАВЛЕНИЯ ВЕРСИЯМИ.

Объём текста не менее 1000 символов! Пока счётчики выставляются в ручную, при необходимости правьте cls-файл.

ABSTRACT

The amount of work sheets 22

Number of images 6

Number of sources used 5

Number of tables 6

COMPUTER SIMULATION, DISTRIBUTED VERSION CONTROL.

Большой текст на английском!

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность

Цель

Задачи

1. Текст много текста, очень много текста. Текст много текста, очень много текста.
2. Текст много текста, очень много текста. Текст много текста, очень много текста. Текст много текста, очень много текста.
3. Текст много текста, очень много текста. Текст много текста, очень много текста. Текст много текста, очень много текста. Текст много текста, очень много текста. Текст много текста, очень много текста. Текст много текста, очень много текста.

ЗАКЛЮЧЕНИЕ

1. Пример ссылки на литературу [1].
2. Пример ссылки на литературу [2].
3. Пример ссылки на литературу [3].
4. Пример ссылки на литературу [4].
5. Пример ссылки на литературу [5].
6. Пример ссылки на литературу [6].

Всего рисунков в документе 0.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Фамилия И.О. Название книги. — Город издательства: «Издательство», Год. — Количество страниц.
2. МакГрат М. Программирование на C для начинающих / пер. с англ. Михаил Райтман. — М: «Эксмо», 2016. — 193.
3. Программирование на языке C++ в среде Qt Creator / Е.Р. Алексеев, Г.Г. Злобин, Д.А. и др. — М.: Альт Линукс, 2015. 448 с.
4. Bitbucket [Электронный ресурс] Википедия — свободная энцциклопедия. Режим доступа: https://ru.wikipedia.org/wiki/Bitbucket. — (Дата обр. 18.06.2023).
5. Github [Электронный ресурс] Википедия — свободная энцциклопедия. Режим доступа: https://ru.wikipedia.org/wiki/GitHub. — (Дата обр. 18.06.2023).
6. Id Software [Электронный ресурс] Википедия — свободная энцциклопедия. Режим доступа: https://ru.wikipedia.org/wiki/Id_Software. — (Дата обр. 18.06.2023).

ПРИЛОЖЕНИЕ

Текст программы

Листинг 1

Исходный код расшифровки Audio CAPTCHA

```
'''Файл с классаом для решения audiocaptcha'''
import speech_recognition as sr
import subprocess
import logger
import os


logger = logger.ConfigLogger(__name__)

class AudioCaptchaSolver():
    '''Класс решателя audio captcha'''

    def __init__(self):
        '''Конструктор класса'''
        # Создаем объект распознавателя речи
        self.recognizer = sr.Recognizer()
        # Распознанное текстовое сообщение
        self.text_message = None


    def recognition_audio(self, path_to_audio: str) -> str:
        '''
        Метод распознавания аудиофайла
        Файлы сохраняются в формате mp3 (обычно содержат шум, кроме мест, где слышен голос)
        '''

        # Преобразование mp3 файла в формат, который подходит для распознавания
        mp3_file = path_to_audio
        wav_file = './audio/audiocaptcha.wav'

        if os.name == 'nt':
            subprocess.run(['C:/ffmpeg/bin/ffmpeg.exe', '-i', mp3_file, wav_file])
        else:
            subprocess.run(['ffmpeg', '-i', mp3_file, wav_file])

        try:
            # Загружаем аудио файл
            audio_captcha = sr.AudioFile(wav_file)

            # Распознаем речь из аудио файла
            with audio_captcha as voice:
                audio_data = self.recognizer.record(voice)
                text_message = self.recognizer.recognize_google(audio_data, language='en-US')
            logger.log_info('Распознавание речи завершено успешно!')
        except Exception as e:
```

```
        logger.log_warning(f'Распознавание завершилось с ошибкой: {e}')

    if text_message:
        self.text_message = text_message
        os.remove(mp3_file)
        os.remove(wav_file)

    return self.text_message
```

Листинг 2

Исходный код автоматизированного решения Audio CAPTCHA

```
'''Это основной файл проекта, в котором будут вызываться классы и методы для решения всех популярных видов captcha'''
from selenium import webdriver
from selenium.webdriver.remote.webdriver import WebDriver
from selenium.webdriver.common.by import By

from random import randint
import time
import requests
import os

from audiocaptcha import AudioCaptchaSolver


class MainWorker():
    '''
    Основной класс проекта, который управляет вызовом дочерних классов для решения определенных видов captcha
    На начальном этапе здесь также будет все, что касается получения captcha с веб-страницы
    '''

    def __init__(self, browser: WebDriver):
        '''Конструктор класса'''
        super().__init__()
        self.browser = browser


    def get_captcha(self, link: str) -> str:
        '''Метод получения captcha со страницы'''
        # Проходим по ссылке
        self.browser.get(link)
        time.sleep(randint(3, 5))

        # Переключаемся на фрейм с чекбоксом captcha
        self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '//*[@id="g-recaptcha"]/div/div/iframe'))
        # Кликаем по чекбоксу "Я не робот"
        self.browser.find_element(By.XPATH, '/html/body/div[2]/div[3]/div[1]/div/div/span').click()
        time.sleep(randint(3, 5))

        # Переключаемся на обычную web-страницу
        self.browser.switch_to.default_content()
        # Переключаемся на фрейм с картинкой captcha
        self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '/html/body/div[2]/div[4]/iframe'))
        # Кликаем на кнопку для перехода к audiocaptcha
```

```
43          self.browser.find_element(By.XPATH, '//*[@id="recaptcha-audio-button"]').click()
44          time.sleep(randint(3, 5))
45
46          # Переключаемся на обычную web-страницу
47          self.browser.switch_to.default_content()
48          # Переключаемся на фрейм с айдиозаписью
49          self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '/html/body/div[2]/div[4]/iframe'))
50          # Находим элемент, содержащий ссылку на аудиозапись
51          audio = self.browser.find_element(By.XPATH, '//*[@id="audio-source"]').get_attribute('src')
52          # Делаем запрос для получения файла
53          response = requests.get(audio)
54          response.raise_for_status()
55
56          # Создаем папку для хранения временных файлов
57          if not os.path.isdir('./audio'):
58              os.mkdir('./audio')
59          path_to_file = './audio/audiocaptcha.mp3'
60          # Сохраняем файл
61          with open(f'{path_to_file}', 'wb') as audioCaptcha:
62              audioCaptcha.write(response.content)
63
64          return path_to_file
65
66
67      def paste_response(self, response_message):
68          '''Метод для вставки результата распознавания'''
69          browser.find_element(By.XPATH, '//*[@id="audio-response"]').send_keys(f'{response_message}')
70          time.sleep(randint(3, 5))
71          browser.find_element(By.XPATH, '//*[@id="recaptcha-verify-button"]').click()
72
73
74  if __name__ == '__main__':
75      '''Запуск программы'''
76      list_of_links = [
77          'https://rucaptcha.com/demo/recaptcha-v2',
78          'https://lessons.zennolab.com/captchas/recaptcha/v2_simple.php?level=low',
79          'https://lessons.zennolab.com/captchas/recaptcha/v2_simple.php?level=middle',
80          'https://lessons.zennolab.com/captchas/recaptcha/v2_simple.php?level=high',
81          'https://lessons.zennolab.com/captchas/recaptcha/v2_nosubmit.php?level=low',
82          'https://lessons.zennolab.com/captchas/recaptcha/v2_nosubmit.php?level=middle',
83          'https://lessons.zennolab.com/captchas/recaptcha/v2_nosubmit.php?level=high',
84          'https://lessons.zennolab.com/ru/advanced'
85      ]
86
87      for link in list_of_links:
88          # Настройки user agent
89          USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0
            ↪  Safari/537.36"
90
91          select_browser = randint(1, 10)
92
93          # Выбор браузера и опций характерных для него
94          if select_browser < 5:
95              options = webdriver.ChromeOptions()
96          else:
```

```
        options = webdriver.EdgeOptions()

    options.add_experimental_option("excludeSwitches", ["enable-automation"])
    options.add_experimental_option('useAutomationExtension', False)
    options.add_argument(f"user-agent={USER_AGENT}")
    options.add_argument("--disable-blink-features=AutomationControlled")

    # Передача параметров
    if select_browser < 5:
        browser = webdriver.Chrome(options=options)
    else:
        browser = webdriver.Edge(options=options)
    browser.implicitly_wait(30)

    # Создаем аудиофайл по указанному пути с captcha
    solver = MainWorker(browser)
    path_to_audio = solver.get_captcha(link)

    # Запускаем распознавание
    captcha_solver = AudioCaptchaSolver()
    response = captcha_solver.recognition_audio(path_to_audio)

    # Вставляем результат распознавания в поле ввода
    solver.paste_response(response)
    time.sleep(randint(10, 15))
```

Листинг 3

Исходный код генератора синтетических CAPTCHA

```
from captcha.image import ImageCaptcha

from random import randint, shuffle
import numpy as np
import os

from textcaptcha.preprocessing_image import preprocessing_image


def generate_image(path_to_file: str, alphabet: list, number_of_start:int, number_of_captcha: int, size_of_image: tuple) -> list:
    # Генерация текстовых captcha
    text = ImageCaptcha(size_of_image[0], size_of_image[1], ['./fonts/arial.ttf', './fonts/comic.ttf', './fonts/cour.ttf', './fonts/georgia.ttf'])
    # Структура возвращаемого списка: [filename, label, (width, height)]
    filenames = []
    for _ in range(number_of_start, number_of_captcha):
        captcha_text = [alphabet[randint(0, len(alphabet) - 1)] for _ in range(randint(4, 7))]
        shuffle(captcha_text)
        text.write(''.join(captcha_text), f'{path_to_file}/{""".join(captcha_text)}.png')
        filenames.append(
            [f'{path_to_file}/{""".join(captcha_text)}.png',
            ''.join(captcha_text)]
        )

    return filenames

```

```

if __name__ == '__main__':
    # Алфавит допустимых символов
    alphabet = 'ABCDEFGHJKLMNPQRSTWXYZ023456789'
    # Создание директории для хранения полноценных синтетических текстовых captcha
    path_to_dataset = '../datasets/captcha'
    if not os.path.isdir(path_to_dataset):
        os.mkdir(path_to_dataset)
    # Создаем датасет из нужного полноценных синтетических captcha длиной от 4 до 7 символов размером 250x60
    filenames = generate_image(path_to_dataset, list(alphabet), 0, 100000, (250, 60))

    # Предобработка изображений
    preprocessing_image(filenames)

    # Для отладки без создания датасета с нуля
    numpy_data = np.array(filenames, dtype=object)
    np.save('data.npy', numpy_data)
```

Листинг 4

Исходный код для предобработки изображений датасета

```
import cv2
import numpy as np


def preprocessing_image(list_filenames: list):
    '''Функция для предобработки изображений или изображения для предсказания'''
    # Предобработка изображений с CAPTCHA
    for file in list_filenames:
        # Открытие изображения в градациях серого
        gray_image = cv2.imread(file[0], 0)
        # Приведение всех изображений к одному размеру ширина х высота)
        resized_image = cv2.resize(gray_image, (250, 60))

        # Морфологический фильтр (дилатация) для сужения символов и более четкого отделения их друг от друга
        morph_kernel = np.ones((3, 3))
        dilatation_image = cv2.dilate(resized_image, kernel=morph_kernel, iterations=1)

        # Применяем пороговую обработку, чтобы получить только черные и белые пиксели
        _, thresholder = cv2.threshold(
            dilatation_image,
            0,
            255,
            cv2.THRESH_BINARY + cv2.THRESH_OTSU
        )

        cv2.imwrite(file[0], thresholder)
```

Листинг 5

Исходный код для создания датасета в формате тензоров

```
import pandas as pd
```

```python
import tensorflow as tf
from keras._tf_keras.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split


def parse_data(image_path: list, encoder_labels: list, decoder_labels: list) -> tuple[tf.Tensor, list]:
    '''Функция для склеивания изображений и лейблов для датасета'''
    image = tf.io.read_file(image_path)
    image = tf.image.decode_png(image, channels=1)
    image = tf.cast(image, tf.float32) / 255.0

    return (image, encoder_labels), decoder_labels


def create_dataset(images: list, encoder_labels: list, decoder_labels: list, shuffle = True, batch_size = 32) -> tf.data.Dataset:
    '''Функция для создания датасета, понятного для TensorFlow'''
    dataset = tf.data.Dataset.from_tensor_slices((images, encoder_labels, decoder_labels))
    dataset = dataset.map(lambda x, y, w: parse_data(x, y, w))
    if shuffle == True:
        dataset = dataset.shuffle(len(images)).batch(batch_size)
    else:
        dataset = dataset.batch(batch_size)

    return dataset


def create_dataframe(images: list) -> pd.DataFrame:
    '''Функция для создания датафреймов на основе списков'''

    # Создание файла с лейблами о содержимом изображений с CAPTCHA
    filenames = [objects[0] for objects in images]
    list_labels = [objects[1] for objects in images]

    # Создание DataFrame для сохранения соотвествия между путями, лейблами и размерами для каждого элемента датасета
    data = {
        'filename': filenames,
        'label': list_labels,
    }

    return pd.DataFrame(data)


def preparing_dataset(dataframe: pd.DataFrame, alphabet: str, shuffle = True) -> tuple[
        tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list],
        tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list]
    ]:
    '''Подготовка датасета'''

    # Сохранение отдельных составляющих DataFrame
    X_captcha, y_captcha = dataframe['filename'].tolist(), dataframe['label'].tolist()

    dict_alphabet = {alphabet[i]:i for i in range(len(alphabet))}
    start_token = len(alphabet)  # Индекс токена <start>
    end_token = len(alphabet) + 1  # Индекс токена <end>

```

```
    # Кодируем лейблы с добавлением токена <start> для кодера
    encoder_labels = [[start_token] + [dict_alphabet[char] for char in label] for label in y_captcha]

    # Кодируем лейблы с добавлением токена <end> для декодера
    decoder_labels = [[dict_alphabet[char] for char in label] + [end_token] for label in y_captcha]

    # Преобразование меток в тензоры
    encoder_tensors = pad_sequences(encoder_labels, maxlen=8, padding='post')
    decoder_tensors = pad_sequences(decoder_labels, maxlen=8, padding='post')

    # Создание датасета
    dataset = create_dataset(X_captcha, encoder_tensors, decoder_tensors, shuffle)

    return dataset


def create_dataset_for_captcha(filenames: list, alphabet: str) -> tuple[
        tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list],
        tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list]
    ]:
    '''Функция для создания датасета на основе алфавита и имен файлов'''

    # Создание датафрейма для удобства последующей обработки
    captcha_dataframe = create_dataframe(filenames)

    # Разделение датасета на обучающую и тестовую выборки
    train_captcha_df, test_captcha_df = train_test_split(captcha_dataframe, test_size=0.2, random_state=42)
    # Разделение тестовой части датасета на валидационную и тестовую выборки
    val_captcha_df, test_captcha_df = train_test_split(test_captcha_df, test_size=0.5, random_state=42)

    train_dataset = preparing_dataset(train_captcha_df, alphabet)
    val_dataset = preparing_dataset(val_captcha_df, alphabet)
    test_dataset = preparing_dataset(test_captcha_df, alphabet, False)

    return train_dataset, val_dataset, test_dataset
```

Листинг 6

Исходный код CRNN модели

```
import tensorflow as tf
import numpy as np
from keras._tf_keras.keras.layers import Input, Conv2D, MaxPooling2D, Reshape, Dense, Dropout, Bidirectional, GRU,
    ↪ BatchNormalization
from keras._tf_keras.keras.regularizers import l2
from keras._tf_keras.keras.models import Model
from keras._tf_keras.keras import backend as K
from keras._tf_keras.keras.backend import ctc_decode
from keras._tf_keras.keras.optimizers import Adam
from keras._tf_keras.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras._tf_keras.keras.saving import register_keras_serializable


def decode_predictions(preds, max_length, alphabet):
    # Используем CTC-декодирование для предсказаний
```

```
    decoded_preds, _ = ctc_decode(preds, input_length=np.ones(preds.shape[0]) * preds.shape[1])
    texts = []
    for seq in decoded_preds[0]:
        text = ''.join([alphabet[i] for i in seq.numpy() if i != -1])  # Исключаем 'blank' символы
        texts.append(text)
    return texts


def decode_batch_predictions(pred):
    # CTC decode
    decoded, _ = ctc_decode(pred, input_length=np.ones(pred.shape[0]) * pred.shape[1],
                            greedy=True)
    decoded_texts = []

    # Преобразование в текст
    for seq in decoded[0]:
        text = ''.join([chr(x) for x in seq if x != -1])  # Пропускаем -1 (пустые символы CTC)
        decoded_texts.append(text)
    return decoded_texts


# Функция CTC Loss
# Функция для декодирования предсказаний модели
@register_keras_serializable(package='Custom', name='ctc_loss')
def ctc_loss(y_true, y_pred):
    # Формируем входные данные для CTC
    input_lenght = tf.ones(shape=(tf.shape(y_pred)[0], 1)) * tf.cast(tf.shape(y_pred)[1], tf.float32)
    label_length = tf.ones(shape=(tf.shape(y_true)[0], 1)) * 7
    return tf.reduce_mean(K.ctc_batch_cost(y_true, y_pred, input_lenght, label_length))


# Модель
def build_model(num_of_classes):
    '''Создание модели'''
    # Входной слой
    input_layer = Input((60, 250, 1))

    # Первый сверточный блок
    x = Conv2D(32, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.003))(input_layer)
    x = BatchNormalization()(x)
    x = Conv2D(32, (3, 3), activation='relu', kernel_regularizer=l2(0.003))(x)
    x = MaxPooling2D((1, 2))(x)
    x = Dropout(0.25)(x)  # Dropout после каждого блока

    # Второй сверточный блок
    x = Conv2D(64, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.003))(x)
    x = BatchNormalization()(x)
    x = Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.003))(x)
    x = MaxPooling2D((1, 2))(x)
    x = Dropout(0.3)(x)

    # Третий сверточный блок
    x = Conv2D(128, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.003))(x)
    x = BatchNormalization()(x)
    x = Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.003))(x)
```

```
    x = MaxPooling2D((1, 2))(x)
    x = Dropout(0.4)(x)

    # Изменяем размерность тензора
    x = Reshape((-1, x.shape[-1] * x.shape[-2]))(x)

    # Первый рекурентный блок
    x = Bidirectional(GRU(128, return_sequences=True))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.6)(x)

    # Второй рекурентный блок
    x = Bidirectional(GRU(128, return_sequences=True))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.6)(x)

    # Третий рекурентный блок
    x = Bidirectional(GRU(128, return_sequences=True))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.6)(x)

    # Выходной слой
    outputs = Dense(num_of_classes + 1, activation='softmax')(x)

    # Создание модели
    model = Model(inputs=input_layer, outputs=outputs)

    return model


def fit_crnn(num_of_classes, train, val):
    # Компиляция модели
    model = build_model(num_of_classes)
    optimizer = Adam(learning_rate=0.001, weight_decay=1e-6)
    model.compile(
        loss=ctc_loss,
        optimizer=optimizer
    )

    # Вывод структуры модели
    model.summary()

    lr_sheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)
    early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    # Обучение модели
    history = model.fit(
        train,
        validation_data=val,
        epochs=15,
        callbacks=[early_stop, lr_sheduler]
    )

    model.save('crnn_model.keras')

```

```
125     return model, history
```

Листинг 7

Исходный код Seq-to-Seq модели

```
1  import tensorflow as tf
2  from keras._tf_keras.keras import layers, Model
3  from keras._tf_keras.keras.callbacks import EarlyStopping, ReduceLROnPlateau
4
5  from create_dataset import create_dataset_for_captcha
6
7
8  # Обновлённый кодировщик
9  def build_encoder():
10     encoder_inputs = layers.Input(shape=(60, 250, 1), name="encoder_inputs")
11
12     # Первый сверточный блок
13     x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoder_inputs)
14     x = layers.BatchNormalization()(x)
15     x = layers.Conv2D(32, (3, 3), activation='relu')(x)
16     x = layers.MaxPooling2D((2, 2))(x)
17
18     # Второй сверточный блок
19     x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
20     x = layers.BatchNormalization()(x)
21     x = layers.Conv2D(64, (3, 3), activation='relu')(x)
22     x = layers.MaxPooling2D((2, 2))(x)
23
24     # Третий сверточный блок
25     x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
26     x = layers.BatchNormalization()(x)
27     x = layers.Conv2D(128, (3, 3), activation='relu')(x)
28     x = layers.MaxPooling2D((2, 2))(x)
29
30     # Четвертый сверточный блок
31     x = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x)
32     x = layers.BatchNormalization()(x)
33     x = layers.Conv2D(256, (3, 3), activation='relu')(x)
34     x = layers.MaxPooling2D((2, 2))(x)
35
36     x = layers.GlobalAveragePooling2D()(x)
37     x = layers.Dense(256, activation="relu")(x)
38     x = layers.BatchNormalization()(x)
39     x = layers.Reshape((1, 256))(x)  # Добавляем временное измерение
40
41     # RNN слой
42     encoder_output, encoder_state = layers.GRU(256, return_sequences=True, return_state=True)(x)
43
44     return Model(encoder_inputs, [encoder_output, encoder_state], name="encoder")
45
46
47 # Декодировщик с Attention
48 def build_decoder(alphabet_size):
49     decoder_inputs = layers.Input(shape=(None,), name="decoder_inputs")
```

```
    encoder_state_input = layers.Input(shape=(256,), name="encoder_state_input")

    x = layers.Embedding(alphabet_size, 128)(decoder_inputs)
    rnn_output, decoder_state = layers.GRU(256, return_sequences=True, return_state=True)(x, initial_state=encoder_state_input)

    # Attention
    attention_output = layers.AdditiveAttention()([rnn_output, encoder_state_input])
    x = layers.Concatenate()([rnn_output, attention_output])
    decoder_outputs = layers.Dense(alphabet_size, activation="softmax")(x)

    return Model([decoder_inputs, encoder_state_input], [decoder_outputs, decoder_state], name="decoder")


def fit_seq_to_seq(number_of_classes: int, train_dataset: tf.data.Dataset, val_dataset: tf.data.Dataset) -> tuple[Model, dict]:
    # Построение полной модели
    encoder = build_encoder()
    decoder = build_decoder(number_of_classes + 2)

    # Полная модель
    encoder_inputs = encoder.input
    decoder_inputs = layers.Input(shape=(None,), name="decoder_inputs")

    _, encoder_state = encoder(encoder_inputs)
    decoder_output, _ = decoder([decoder_inputs, encoder_state])

    seq2seq_model = Model([encoder_inputs, decoder_inputs], decoder_output, name="seq2seq_model")

    # Компиляция модели
    seq2seq_model.compile(
        loss="sparse_categorical_crossentropy",
        optimizer="adam",
        metrics=["accuracy"]
    )

    seq2seq_model.summary()

    lr_sheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)
    early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    # Обучение модели
    history = seq2seq_model.fit(
        train_dataset,
        validation_data=val_dataset,
        epochs=20,
        callbacks=[early_stop, lr_sheduler]
    )

    seq2seq_model.save('seq_to_seq_model.keras')

    return seq2seq_model, history
```

Листинг 8

Исходный код тестирования модели

```python
import numpy as np
import tensorflow as tf
from keras._tf_keras.keras.models import load_model


if __name__ == '__main__':
    import matplotlib.pyplot as plt
    import seaborn as sbn

    # Алфавит допустимых символов
    alphabet = ' ABCDEFGHJKLMNPQRSTWXYZ023456789'

    list_filenames = np.load('data.npy', allow_pickle=True)
    # Создание единого датасета
    captcha_dataset = create_dataset_for_captcha(list_filenames, alphabet)
    if False:
        # Обучение модели
        model_captcha, history_captcha = fit_seq_to_seq(len(alphabet), captcha_dataset[0], captcha_dataset[1])
        # Построение графика изменения val_loss и loss
        plt.plot(history_captcha.history['loss'], label='Training Loss')
        plt.plot(history_captcha.history['val_loss'], label='Validation Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()
        # Сохраняем график для отчета
        plt.savefig('Model_loss.png')

    # Загружаем предобученную модель и получаем предсказания для тестовой выборки
    model = load_model('seq_to_seq_model.keras')
    predictions = model.predict(captcha_dataset[2])

    # Переводим предсказания из представления вероятностей в классы
    pred_classes = np.argmax(predictions, axis=-1)
    captcha_labels = [label.numpy() for _, label in captcha_dataset[2].unbatch()]
    captcha_labels = np.array(captcha_labels)

    # Убираем padding
    def remove_padding(sequences, padding_value=0):
        return [seq[seq != padding_value] for seq in sequences]

    # Убираем padding из предсказаний и меток
    pred_classes_no_padding = remove_padding(pred_classes, padding_value=0)
    true_labels_no_padding = remove_padding(np.array(captcha_labels), padding_value=0)

    # Проверяем размеры списков после удаления padding
    print(f"Количество предсказаний: {len(pred_classes_no_padding)}")
    print(f"Количество меток: {len(true_labels_no_padding)}")

    # Проверяем совпадение предсказаний и истинных меток посимвольно
    sequence_accuracy = np.mean(
        [np.array_equal(pred, true) for pred, true in zip(pred_classes, captcha_labels)]
    )
    print(f"Точность последовательностей (без padding): {sequence_accuracy:.4f}")

    # Расчет точности символов (character-level accuracy)
```

```
total_characters = np.prod(captcha_labels.shape)
correct_characters = np.sum(pred_classes == captcha_labels)
character_accuracy = correct_characters / total_characters
print(f"Точность символов: {character_accuracy:.4f}")

from sklearn.metrics import confusion_matrix
# Построение матрицы ошибок для анализа
true_symb, pred_symb = [], []

for true_seq, pred_seq in zip(true_labels_no_padding, pred_classes_no_padding):
    true_symb.extend(true_seq)
    pred_symb.extend(pred_seq)
cm = confusion_matrix(true_symb, pred_symb)

plt.figure(figsize=(10, 7))
sbn.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted classes')
plt.ylabel('True classes')
plt.title('Confusion matrix')
# plt.show()
plt.savefig('Confusion_matrix.png')

from collections import defaultdict

sequence_accuracy_by_length = defaultdict(list)
for pred, true in zip(pred_classes_no_padding, true_labels_no_padding):
    seq_len = len(true)
    is_correct = np.array_equal(pred, true)
    sequence_accuracy_by_length[seq_len].append(is_correct)

# Считаем точность для каждой длины
for length, results in sequence_accuracy_by_length.items():
    acc = np.mean(results)
    print(f"Длина {length}: Точность {acc:.4f}")
```

Листинг 9

Исходный код получения CAPTCHA с целевого сайта

```
# Подключение библиотек для работы с браузером
from selenium import webdriver
from selenium.webdriver.remote.webdriver import WebDriver
from selenium.webdriver.common.by import By

# Подключение библиотек для работы с текстом заданиия captcha
from deep_translator import GoogleTranslator
import inflect

# Библиотека для парсинга HTML
from bs4 import BeautifulSoup

from random import randint
import time
import requests
import os
```

```
17 import csv
18
19
20 class GetCaptcha():
21     '''
22     Основной класс проекта, который управляет вызовом дочерних классов для решения определенных видов captcha
23     На начальном этапе здесь также будет все, что касается получения captcha с веб-страницы
24     '''
25
26     def __init__(self, browser: WebDriver):
27         '''Конструктор класса'''
28         super().__init__()
29         self.browser = browser
30
31
32     def get_captcha(self, link: str, cnt: int) -> tuple[str, str, str]:
33         '''Метод получения captcha со страницы'''
34         # Проходим по ссылке
35         self.browser.get(link)
36         time.sleep(randint(3, 5))
37
38         # Переключаемся на фрейм с чекбоксом captcha
39         self.browser.switch_to.frame(self.browser.find_element(
40             By.XPATH,
41             '//*[@id="g-recaptcha"]/div/div/iframe'
42         ))
43         # Кликаем по чекбоксу "Я не робот"
44         self.browser.find_element(
45             By.XPATH,
46             '/html/body/div[2]/div[3]/div[1]/div/div/span'
47         ).click()
48         time.sleep(randint(3, 5))
49
50         # Переключаемся на обычную web-страницу
51         self.browser.switch_to.default_content()
52         # Переключаемся на фрейм с картинкой captcha
53         self.browser.switch_to.frame(self.browser.find_element(
54             By.XPATH,
55             '/html/body/div[2]/div[4]/iframe'
56         ))
57         # Находим элемент, содержащий ссылку на исходное изображение
58         image = self.browser.find_element(
59             By.XPATH,
60             '//*[@id="rc-imageselect-target"]/table/tbody'+
61             '/tr[1]/td[1]/div/div[1]/img'
62         ).get_attribute('src')
63         # Делаем запрос для получения файла
64         response = requests.get(image)
65         response.raise_for_status()
66
67         # Получаем название объекта, который надо найти
68         object_name = self.browser.find_element(
69             By.XPATH,
70             '//*[@id="rc-imageselect"]/div[2]/div[1]/div[1]'+
71             '/div/strong'
```

```
72          ).text
73
74          # Получаем таблицу с кусочками изображения
75          table = self.browser.find_element(
76              By.XPATH,
77              '//*[@id="rc-imageselect-target"]/table'
78          ).get_attribute('outerHTML')
79
80          # Создаем папку для хранения временных файлов
81          if not os.path.isdir('../datasets/imagecaptcha_dataset'):
82              os.mkdir('../datasets/imagecaptcha_dataset')
83          path_to_file = f'../datasets/imagecaptcha_dataset/{cnt}.jpg'
84          # Сохраняем файл
85          with open(f'{path_to_file}', 'wb') as imageCaptcha:
86              imageCaptcha.write(response.content)
87
88          return object_name, path_to_file, table
89
90
91      def get_number_of_cells(self, table:str) -> tuple[int, int]:
92          '''Метод для получения колличества ячеек таблицы для последующего разбиения изображения на части'''
93          # Парсинг HTML
94          soup = BeautifulSoup(table, 'lxml')
95
96          # Получаем количество строк
97          number_of_rows = len(soup.find_all('tr'))
98
99          # Получаем количество столбцов
100         number_of_columns = len(soup.find('tr').find_all(['td', 'th']))
101
102         return number_of_rows, number_of_columns
103
104
105 if __name__ == "__main__":
106     # Целевой сайт
107     target_link = 'https://rucaptcha.com/demo/recaptcha-v2'
108     for cnt in range(463, 638):
109         # Настройки user agent
110         USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0
            ↪  Safari/537.36"
111         options = webdriver.ChromeOptions()
112
113         options.add_experimental_option("excludeSwitches", ["enable-automation"])
114         options.add_experimental_option('useAutomationExtension', False)
115         options.add_argument(f"user-agent={USER_AGENT}")
116         options.add_argument(
117             "--disable-blink-features=AutomationControlled"
118         )
119
120         # Передача параметров
121         browser = webdriver.Chrome(options=options)
122         browser.implicitly_wait(30)
123
124         captcha = GetCaptcha(browser)
125         # Получение captcha и объекта для поиска
```

```
126     task_object, image, table = captcha.get_captcha(target_link, cnt)
127
128     # Перевод названия объекта на английский и сохранение его в единственном числе
129     task_object = GoogleTranslator(source='auto', target='en').translate(task_object)
130     singular = inflect.engine()
131     if len(task_object) > 3:
132         # Исключаем ошибки с множественным числом для слов, которые не могут быть во множественном числе из-за малого
            ↪ количества символов
133         task_object = singular.singular_noun(task_object)
134         if task_object.lower() == 'hydrant':
135             task_object = 'fire hydrant'
136
137     # Получаем количество ячеек
138     rows, columns = captcha.get_number_of_cells(table)
139
140     # Запись полученных параметров в csv-файл
141     with open('images_for_captcha.csv', 'a') as datasetFile:
142         csv_rows = csv.writer(datasetFile, quoting=csv.QUOTE_NONE)
143         csv_rows.writerow([task_object, image, rows, columns])
```

Листинг 10

Исходный код дообучения модели на датасете

```
1  from ultralytics import YOLO
2
3  # Загрузка модели
4  model = YOLO("yolov8m-seg.pt")  # Загрузка предобученной лёгкой модели
5  # Дообучение модель
6  model.train(
7      data="../datasets/image_dataset/image_captcha.yaml",  # Путь к файлу конфигурации
8      epochs=35,
9      imgsz=640,
10     batch=8,
11     workers=4,
12     device="cpu",
13     name="captcha_seg"  # Название директории для сохранения результатов обучения
14 )
```

Листинг 11

Исходный код автоматизированного решения CAPTCHA

```
1  # Подключение библиотек для работы с браузером
2  from selenium import webdriver
3  from selenium.webdriver.remote.webdriver import WebDriver
4  from selenium.webdriver.common.by import By
5  from selenium.common.exceptions import ElementClickInterceptedException
6
7  # Подключение библиотек для работы с текстом задания captcha
8  from deep_translator import GoogleTranslator
9  import inflect
10
11 # Библиотека для парсинга HTML
```

```
from bs4 import BeautifulSoup

# Библиотека для работы с изображениями
from ultralytics import YOLO
import cv2
import numpy as np

from random import randint
import time
import requests


class SolveCaptcha():
    '''
    Основной класс проекта, который управляет вызовом дочерних классов для решения определенных видов captcha
    На начальном этапе здесь также будет все, что касается получения captcha с веб-страницы
    '''

    def __init__(self, browser: WebDriver):
        '''Конструктор класса'''
        super().__init__()
        self.browser = browser


    def find_captcha(self, link: str):
        # Проходим по ссылке
        self.browser.get(link)
        time.sleep(randint(3, 5))

        # Переключаемся на фрейм с чекбоксом captcha
        self.browser.switch_to.frame(self.browser.find_element(
            By.XPATH,
            '//*[@id="g-recaptcha"]/div/div/iframe'
        ))
        # Кликаем по чекбоксу "Я не робот"
        self.browser.find_element(By.XPATH, '/html/body/div[2]/div[3]/div[1]/div/div/span').click()
        time.sleep(randint(3, 5))

        # Переключаемся на обычную web-страницу
        self.browser.switch_to.default_content()
        # Переключаемся на фрейм с картинкой captcha
        self.browser.switch_to.frame(self.browser.find_element(
            By.XPATH,
            '/html/body/div[2]/div[4]/iframe'
        ))


    def get_captcha(self) -> tuple[str, str, str, np.ndarray]:
        '''Метод получения captcha со страницы'''
        # Находим элемент, содержащий ссылку на исходное изображение
        src_image = self.browser.find_element(
            By.XPATH,
            '//*[@id="rc-imageselect-target"]/table/tbody/'+
            'tr[1]/td[1]/div/div[1]/img'
        ).get_attribute('src')
```

```
        # Делаем запрос для получения файла
        response = requests.get(src_image)
        response.raise_for_status()

        # Получаем название объекта, который надо найти
        object_name = self.browser.find_element(
            By.XPATH,
            '//*[@id="rc-imageselect"]/div[2]/div[1]/div[1]/'+
            'div/strong'
        ).text

        # Получаем таблицу с кусочками изображения
        table = self.browser.find_element(
            By.XPATH,
            '//*[@id="rc-imageselect-target"]/table'
        ).get_attribute('outerHTML')

        # Преобразование байтовой последовательности в изображение
        image = cv2.imdecode(np.frombuffer(response.content, np.uint8), cv2.IMREAD_COLOR)

        return object_name, table, src_image, image


    def get_properties_for_recognition(self, task_object: str, table: str) -> tuple[str, int, int]:
        '''Метод для получения необходимых параметров для распознавания на картинке'''
        # Перевод названия объекта на английский и сохранение его в единственном числе
        task_object = GoogleTranslator(source='auto', target='en').translate(task_object)
        singular = inflect.engine()
        if len(task_object) > 3:
            # Исключаем ошибки с множественным числом для слов, которые не могут быть во множественном числе из-за малого
            ↪ количества символов
            task_object = singular.singular_noun(task_object)
            if task_object.lower() == 'hydrant':
                task_object = 'fire hydrant'

        # Парсинг HTML
        soup = BeautifulSoup(table, 'lxml')
        # Получаем количество строк
        number_of_rows = len(soup.find_all('tr'))
        # Получаем количество столбцов
        number_of_columns = len(soup.find('tr').find_all(['td', 'th']))

        return task_object, number_of_rows, number_of_columns


    def predict_class(self, image: np.ndarray, task_object: str) -> list:
        '''Метод для получения масок для изображения с необходимым классом'''
        # Передаем в предобученную модель изображение для поиска нужного объекта
        results = model(image)[0]
        class_names = model.names

        # Получаем идентификатор нужного класса
        for id, name in class_names.items():
            if name == task_object.lower():
                class_id = id
```

```
            break

    # Получаем все маски для классов
    masks = results.masks.data.cpu().numpy()
    classes = results.boxes.cls.cpu().numpy()

    # Получаем список масок для нужного класса
    selected_masks = [masks[i] for i in range(len(classes)) if int(classes[i]) == class_id]

    return selected_masks


def get_cells_with_mask(self, cells_with_object: list, coords_cells: list, mask: np.ndarray, grid_size: tuple, threshold: float) -> list:
    '''Метод для получения ячеек таблицы, содержащих объект'''
    # Определяем размер ячейки
    cell_height, cell_width = int(mask.shape[0] / grid_size[0]), int(mask.shape[1] / grid_size[1])
    idx_cell = 0

    for i in range(grid_size[0]):
        for j in range(grid_size[1]):
            # Координаты прямоугольника, соответствующего ячейке
            y1, y2 = i * cell_height, (i + 1) * cell_height
            x1, x2 = j * cell_width, (j + 1) * cell_width

            # Вырезаем часть маски, соответствующую ячейке
            cell_mask = mask[y1:y2, x1:x2]
            # Рассчитываем какую часть ячейки занимает объект
            coverage_area = np.sum(cell_mask) / cell_mask.size

            # Проверяем, есть ли объект в ячейке
            if coverage_area >= threshold:
                # Сохраняем данные о ячейке
                cells_with_object.append(idx_cell)
                coords_cells.append((i, j))

            idx_cell += 1

    return cells_with_object, coords_cells


if __name__ == "__main__":
    # Загружаем модель
    model = YOLO('best.pt')

    # Целевой сайт
    target_link = 'https://rucaptcha.com/demo/recaptcha-v2'

    # Настройки user agent
    USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0
    ↪ Safari/537.36"
    options = webdriver.ChromeOptions()

    options.add_experimental_option("excludeSwitches", ["enable-automation"])
    options.add_experimental_option('useAutomationExtension', False)
    options.add_argument(f"user-agent={USER_AGENT}")
```

```
options.add_argument(
    "--disable-blink-features=AutomationControlled"
)

# Передача параметров
browser = webdriver.Chrome(options=options)
browser.implicitly_wait(30)

captcha = SolveCaptcha(browser)
# Находим фрейм с captcha (автоматизация клика на чекбокс)
captcha.find_captcha(target_link)

# Выполняем распознавание до тех пор, пока фрейм не исчезнет
while True:
    try:
        # Получение изображения captcha и объекта для поиска
        task_object, table, src_image, image = captcha.get_captcha()
        # Полчаем необходимые параметры captcha
        task_object, rows, columns = captcha.get_properties_for_recognition(task_object, table)

        RECURSIVE_CAPTCHA = True    # Флаг для captcha, в которых вместо выбранных изображений появляются новые
        while RECURSIVE_CAPTCHA:
            # Сбрасываем флаг рекурсии
            RECURSIVE_CAPTCHA = False
            # Находим нужный класс на изображении
            selected_masks = captcha.predict_class(image, task_object)

            cells_with_object, coords = [], []
            for mask in selected_masks:
                # Проходим по выбранным маскам для определения клетки к которой она принадлежит
                resized_mask = cv2.resize(mask, (image.shape[1], image.shape[0]), interpolation=cv2.INTER_NEAREST)
                cells_with_object, coords = captcha.get_cells_with_mask(cells_with_object, coords, resized_mask, (rows, columns), 0.05)

            # Кликаем по ячейкам с уникальными индексами
            for cell, coord in list(set(zip(cells_with_object, coords))):
                captcha.browser.find_elements(By.TAG_NAME, 'td')[cell].click()
                time.sleep(randint(2, 3))
                # Проверяем наличие новых изображений в данной ячейке
                src_cell = captcha.browser.find_elements(By.TAG_NAME, 'td')[cell].find_element(By.TAG_NAME, 'img').get_attribute('src')
                if src_cell != src_image:
                    # Делаем запрос для получения изображения
                    response = requests.get(src_cell)
                    response.raise_for_status()
                    cell_image = cv2.imdecode(
                        np.frombuffer(response.content, np.uint8),
                        cv2.IMREAD_COLOR
                    )

                    # Заменяем в исходном изображении старую ячейку на новую
                    x1, x2 = coord[0] * cell_image.shape[0], (coord[0] + 1) * cell_image.shape[0]
                    y1, y2 = coord[1] * cell_image.shape[1], (coord[1] + 1) * cell_image.shape[1]
                    image[x1:x2, y1:y2] = cell_image

                    # Устанавливаем флаг рекурсии
                    RECURSIVE_CAPTCHA = True
```

```
230
231         # Находим кнопку подтверждения выбора и кликаем по ней
232         captcha.browser.find_element(By.XPATH, '//*[@id="recaptcha-verify-button"]').click()
233         time.sleep(randint(3, 5))
234     except ElementClickInterceptedException:
235         captcha.browser.quit()
236         break
237
```

ПОСЛЕДНИЙ ЛИСТ ВКР

Выпускная квалификационная работа выполнена мной совершенно самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

«___» _____ 2025 г.

_____ А. В. Лаптев