УДК 004.89
DOI 10.18822/byusu20250487-98

ПРОГРАММНЫЙ КОМПЛЕКС АВТОМАТИЧЕСКОЙ ПРОВЕРКИ РУКОПИСНЫХ РАБОТ

Парунов Даниил Алексеевич
преподаватель Инженерной школы
цифровых технологий,
Югорский государственный университет,
Ханты-Мансийск, Россия
E-mail: d_parunov@ugrasu.ru

Сафонов Егор Иванович
кандидат физико-математических наук, доцент,
доцент Инженерной школы цифровых технологий,
Югорский государственный университет,
Ханты-Мансийск, Россия
E-mail: e_safonov@ugrasu.ru

Предмет исследования: время работы преподавателей учебных заведений на этапе проверки рукописных работ (контрольные, самостоятельные работы), а также алгоритмы автоматического распознавания и проверки рукописного текста.

Цель исследования: создание программного комплекса для проверки рукописных работ учащихся общеобразовательных организаций, используя алгоритмы оптического распознавания символов (OCR), машинного обучения и средств веб-разработки.

Объекты исследования: процессы и методы автоматизированной проверки рукописных работ учащихся общеобразовательных организаций с использованием современных технологий распознавания текста и машинного обучения.

Методы исследования: анализ способов и методов распознавания рукописного текста при проведении единого государственного экзамена, обзор современных сервисов и фреймворков OCR трансформерного типа, а также построение архитектуры клиент-серверного веб-приложения.

Основные результаты исследования: построенная архитектура комплекса, включающая модули инициализации модели, предобработки изображений с функциями выравнивания, шумоподавления и сегментации, распознавания текста, агрегации результатов и удобный веб-интерфейс для загрузки работ и просмотра результатов.

Ключевые слова: машинное обучение, GPT, OCR, Python, микросервис.

A SOFTWARE PACKAGE FOR AUTOMATIC CHECKING OF HANDWRITTEN WORKS

Daniil A. Parunov
Lecturer of the Engineering School
of Digital Technologies,
Yugra State University,
Khanty-Mansiysk, Russia
E-mail: d_parunov@ugrasu.ru

Egor I. Safonov
Candidate of Physics and Mathematics,
Associate Professor,
Associate Professor of the Engineering School
of Digital Technologies,
Yugra State University,
Khanty-Mansiysk, Russia
E-mail: e_safonov@ugrasu.ru

Subject of research: the time spent by teachers at educational institutions during the verification of handwritten assignments (tests, independent assignments), as well as algorithms for automatic recognition and verification of handwritten text.

Purpose of research: development a software for verifying handwritten assignments of students in general education organizations using optical character recognition (OCR) algorithms, machine learning, and web development tools.

Objects of research: processes and methods for automated verification of handwritten assignments of students in general education organizations using modern text recognition and machine learning technologies.

Research methods: analysis of handwriting recognition methods and techniques during the Unified State Exam, a review of modern transformer-type OCR services and frameworks, and the construction of a client-server web application architecture.

Research findings: the developed architecture of the software package, including modules for model initialization, image preprocessing with alignment, noise reduction, and segmentation functions, text recognition, results aggregation, and a user-friendly web interface for uploading assignments and viewing results.

Keywords: machine learning, GPT, OCR, Python, microservices.

ВВЕДЕНИЕ

Контроль и оценка усвоения материала обучающимися образовательных учреждений являются основой для управления образовательным процессом, позволяя выявлять уровень знаний и умений обучающихся, определять степень достижения поставленных целей и принимать необходимые корректирующие меры для обеспечения эффективности обучения. Традиционные формы контроля, такие как контрольные и самостоятельные работы, требуют значительных временных затрат преподавателей на проверку, что снижает их возможность заниматься другими важными аспектами педагогической деятельности.

Одним из наиболее перспективных направлений автоматизации образовательного процесса является внедрение систем автоматической проверки рукописных работ. Опыт проведения Единого государственного экзамена (ЕГЭ) демонстрирует успешный переход от частичной к полностью автоматизированной проверке бланков, что подтверждает принципиальную возможность и эффективность данного подхода в масштабах всей страны. Однако существующие государственные системы являются закрытыми и не адаптированы для повседневного использования в школах.

Современные технологии, в частности алгоритмы оптического распознавания символов и машинного обучения, достигли уровня, позволяющего создавать решения для

автоматической проверки школьных работ. А развитие трансформерных архитектур открывает новые возможности для точного распознавания рукописного текста, преодолевая ограничения традиционных систем, которые демонстрируют высокий уровень ошибок при работе с почерком.

Целью данной работы является проектирование и разработка программного комплекса для автоматической проверки рукописных работ учащихся общеобразовательных организаций:

– провести анализ современных технологий, алгоритмов и фреймворков для автоматического распознавания рукописного текста;

– спроектировать реляционную базу данных и микросервисную архитектуру программного комплекса;

– реализовать интеграционную архитектуру и ключевые модули системы.

Разрабатываемый программный комплекс призван сократить временные затраты учителей на рутинную проверку работ, повысить объективность оценивания и позволить педагогам сосредоточиться на индивидуальной работе с учащимися и совершенствовании учебного процесса.

РЕЗУЛЬТАТЫ И ОБСУЖДЕНИЕ

Рассмотрим технологии и алгоритмы автоматического распознавания и проверки рукописного текста. Известно, что при проверке работ ЕГЭ используются алгоритмы оптического распознавания символов (Optical Character Recognition, OCR), но данная система используется только внутри организаций РЦОИ, и доступ извне к ней закрыт.

Алгоритм OCR – это процесс преобразования изображения текста в машиночитаемый текстовый формат. В большинстве случаев применяется простой алгоритм для распознавания символов – сопоставление шаблонов из заранее обученной базы (токенов), который называется конволюцией. Принцип работы OCR строится на алгоритмах нейронных сетей, которые распознают символы и слова наподобие человека. Данные алгоритмы являются частью интеллектуального распознавания символов (Intelligent Character Recognition, ICR). OCR использует «свёрточное ядро» для обнаружения определённого паттерна на изображении и поочерёдно применяет данный паттерн на каждую область пикселей графического файла, где область определяется размерами ядра [4].

Для задачи распознавания рукописных работ была выбрана задача извлечения ключевой информации (Key Information Extraction, KIE). В качестве материала для распознавания был выбран бланк выпускного Единого государственного экзамена для краткой части (рисунок 1).

Рисунок 1. Пример бланка ЕГЭ и неправильного распознавания

В качестве модели для распознавания рукописного текста используется модель Tesseract, которая на текущий момент не совсем корректно справляется с анализом рукописного текста: часто буквы не определяются корректно или случается ошибочное распознавание. Заполнение бланка требует следования жёстким правилам, таким как заполнение только чёрной ручкой или написание букв согласно образцам.

Современные же модели машинного обучения позволяют отойти от строгих правил за счёт большего объёма данных для обучения и улучшения алгоритмов нейронных сетей.

Проводя анализ доступных аналогов, выделим две категории – фреймворки и сервисы. Большинство рассматриваемых сервисов работает по модели SaaS – Software as a Service, предоставляя пользователю лишь программное обеспечение, способное выполнить необходимый запрос по представленным пользователем данным. В то время как фреймворки предоставляют большую гибкость и позволяют пользователю, не прибегая к сторонним сервисам, реализовать лишь часть функционала программного обеспечения [1].

Доступными отечественными и зарубежными инструментами OCR являются:

– Yandex Vision OCR;
– ChatGPT;
– DeepSeek.

Компания «Яндекс» предлагает тарифы с определенными лимитами, предоставляющие достаточно хороший арсенал для распознавания рукописного текста, – Yandex Vision OCR, включающий в себя:

– обращение к API-сервису;
– доступную документацию разработчика;
– поддержку множества «шаблонных» документов;
– возможность выбрать необходимую модель для обработки рукописного или машинопечатного текста.

Главным недостатком данного сервиса является невозможность корректно распознать текст, символы которого записаны в отдельных ячейках, о чём написано в разделе документации [9].

Следующим рассматриваемым продуктом были взяты мультимодальные модели ChatGPT и DeepSeek. Данные продукты изначально разрабатывались как большие лингвистические модели, способные выполнять множество задач, но их также можно использовать в качестве сервиса для распознавания текста.

Модель OpenAI ChatGPT неплохо справляется с поставленной задачей, даже если требуется выполнить дополнительное преобразование текста в определённый формат или язык разметки, например LaTeX.

Основной недостаток данного сервиса – его высокая стоимость. На момент первой половины 2025 года цена подписки для доступа к премиум-версии составляет от 16 до 20 тысяч рублей, а для доступа к API используются расценки за 1 миллион токенов от 500 до 1500 рублей.

В свою очередь, компания DeepSeek в первой половине 2025 года выпустила свою мультимодальную нейронную сесть в открытый доступ, снискав огромную аудиторию и популярность среди обычных пользователей. Модель распространяется открыто, и каждый пользователь, обладающий определёнными навыками системного администрирования, может развернуть данную модель на своём оборудовании.

Стоимость доступа к API у DeepSeek гораздо демократичнее, чем у ChatGPT, всего от 110 до 300 рублей за миллион токенов.

При разработке программного обеспечения стоит учитывать, что при использовании фреймворков, а не сторонних сервисов, вся сложность при обработке данных ложится именно на производительные мощности компьютера, на котором будет запущен данный программный продукт: фреймворк использует локальные вычислительные мощности вместо облачных, что стоит учитывать при разработке сложных и распределённых систем.

Мы не будем рассматривать фреймворк *Tesseract* ввиду устаревания его алгоритма распознавания, что приводит к значительному проценту ошибки распознавания текста [7].

Фреймворк *EasyOCR* позволяет в кратчайшие сроки подготовить модель для распознавания текста, не требующего предварительной обработки. Данный фреймворк использует особую последовательность этапов (pipeline) для распознавания текста, представленную на рисунке 2.

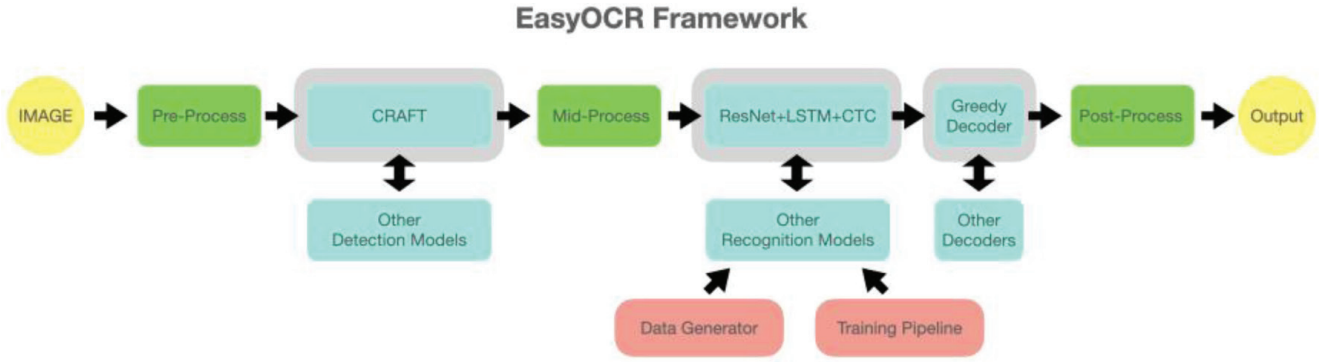

Рисунок 2. Последовательность этапов распознавания текста фреймворка EasyOCR

Данный фреймворк использует графический ускоритель в качестве основного ядра для вычислений, но и имеет поддержку вычислений на процессоре.

Кроме того, фреймворк требует дообучения модели для определённого словаря, но использование русского алфавита не предусмотрено, что потребовало бы дообучить модель и проверять выходные данные на корректность текущему словарю [3].

Основным недостатком данного фреймворка является невозможность извлечения информации по ключам, что необходимо для решения текущей задачи.

Фреймворк *PaddleOCR* – это китайская модель для обнаружения и распознавания текста, предоставляет множество инструментов для разработки, обучения модели и возможности применения в различных сферах. Фреймворк поддерживает множество языков для распознавания текста и позволяет нам извлекать информацию по ключам, что сильно упрощает разработку решения для нашей задачи.

PaddleOCR имеет несколько предобученных моделей, способных работать как на базе графического ускорителя, так и на центральном процессоре, не теряя в скорости вычислений.

Основным недостатком данного фреймворка является низкая обученность для распознавания русского рукописного текста, что может привести к ошибкам.

Фреймворк *Transformers* является инструментом для использования заранее предобученных моделей с общим назначением – возможно использование моделей для генерации текста, генерации изображений, перевода текста в аудио и распознавание текста на изображении. Данный фреймворк не решает определённую задачу, но через него возможно использование различных моделей из открытого доступа.

Среди множества предобученных моделей для распознавания текста есть и необходимые под наши задачи модели, обученные на кириллическом алфавите, например kazars24/trocr-base-handwritten-ru [5].

Для достижения цели можно выделить 3 основных микросервиса (подсистемы):

– веб-интерфейс – создан для взаимодействия пользователя с системой, но не является основным интерфейсом взаимодействия;

– логическая часть (обработчик запросов) – является основным интерфейсом взаимодействия и создана для обработки всех запросов; выполняет основную логику бизнес-задач;

– обработчик рукописных работ – является моделью искусственного интеллекта, способной обработать графический файл и вернуть структурированные данные для дальнейшей обработки.

Функциональные требования к подсистеме веб-интерфейса стандартные – отображение главных страниц.

Зона ответственности подсистемы обработчика запросов достаточно обширна, но является минимально возможной в рамках данного программного продукта. Является также промежуточным микросервисом для связи с обработчиком рукописных работ. Основными функциональными требованиями к подсистеме являются:

– обработка запросов с веб-интерфейса, соответствующих его функциональным требованиям;

– обработка запросов на обработку серии изображений или файла с графическими изображениями;

– обеспечивать взаимосвязь между компонентами остального программного комплекса;

– использовать механизм кэширования данных с помощью NoSQL базы данных типа «ключ-значение» для обеспечения быстродействия подсистемы.

Для подсистемы обработчика рукописного текста были выделены следующие основные функциональные требования:

– подсистема должна принимать входные данные в виде изображения / серии изображений в формате PNG или документа / серии документов в формате PDF;

– подсистема должна анализировать рукописный и/или машинописный текст с входных данных (файлов) и переводить его в формат данных, определённый в процессе проектирования;

– подсистема должна после анализа рукописного и/или машинописного текста с входных данных (файлов) и перевода его в определённый формат данных передавать его для дальнейшей работы в подсистему обработчика запросов;

– подсистема должна иметь возможность принимать вместе с входными данными их разметку для дальнейшего анализа рукописного и/или машинописного текста.

Для работы приложения необходимо спроектировать диаграмму отношений сущностей ERD (рисунок 3), включающую:

– сущность пользователя;
– сущность группы;
– сущность ролей;
– сущность прав ролей;
– сущность работы;
– сущность вопросов работы;
– сущность ответов работ.

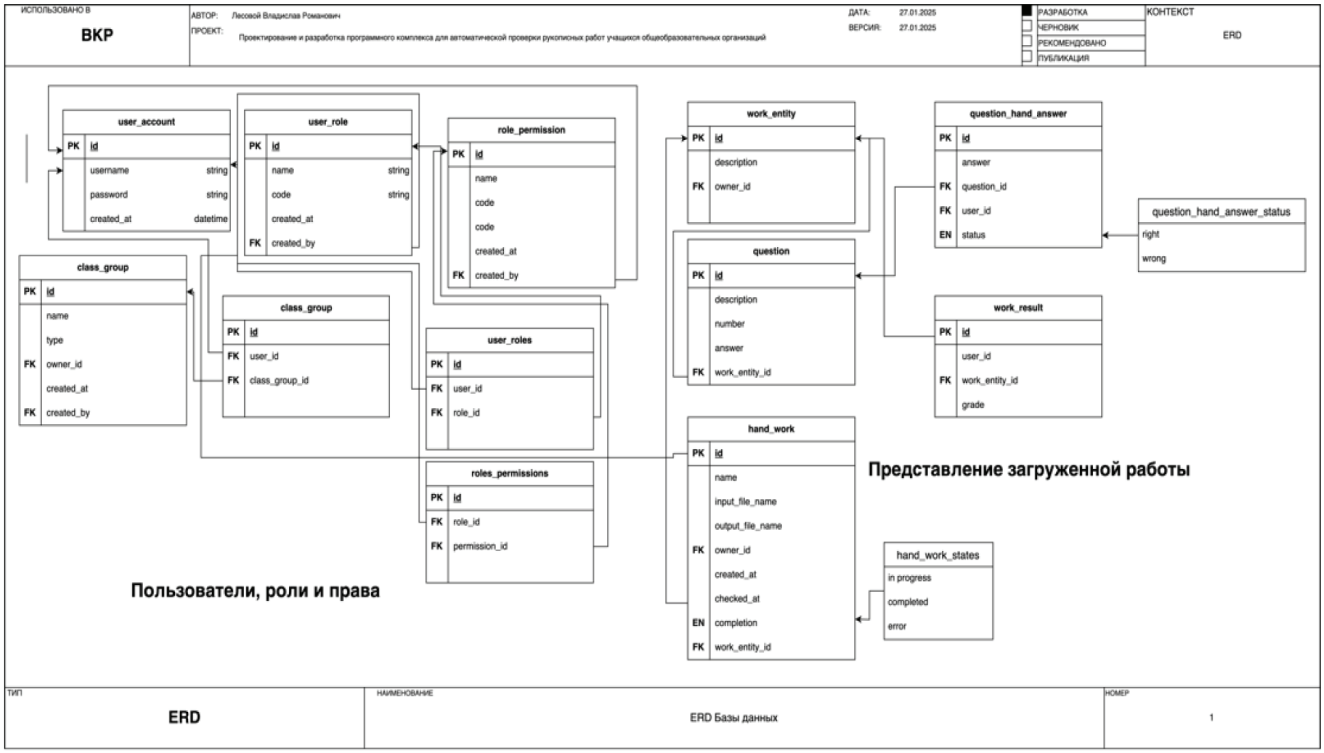

Рисунок 3. Диаграмма отношений сущностей (ERD)

Для разработки микросервиса был выбран язык программирования Python, а в качестве основного фреймворка для работы с веб-запросами был выбран FastAPI, который позволяет с помощью множества встроенных в него библиотек достаточно быстро реализовать необходимый функционал с возможностью дальнейшего масштабирования.

Использование фреймворка FastAPI с применением стандартов ReST позволяет разработать универсальный интерфейс доступа и сократить время разработки в целом. Фреймворк поддерживает возможность получения запросов не только с текстовыми данными в формате JSON, но и получения файлов с помощью смешанного запроса типа multipart/form-data [2].

В качестве второстепенных фреймворков были выбраны Pydantic и SQLAlchemy.

Фреймворк Pydantic позволяет создавать необходимые модели данных и использовать их с применением основных принципов объектно ориентированного программирования. Также существует возможность заранее описать необходимый тип данных, который будет использоваться в модели данных для прохождения валидации и избежания ошибок с заменой одного типа данных на другой. Также имеется поддержка написания собственных валидаторов в момент создания модели, обращения к модели или изменения данных [6].

Фреймворк SQLAlchemy является очень удобной системой ORM для связи с базой данных, позволяя обращаться к данным как к объекту. При создании сущностей SQLAlchemy был использован подход Code-first, который позволяет масштабировать систему и проводить миграции с помощью утилиты Alembic [8].

В рамках решения задачи автоматической проверки бланков с ответами была разработана система на основе модели машинного обучения TrOCR, предназначенной для распознавания рукописного текста. Система включает этапы предобработки изображений, выделения областей интереса, распознавания текста и постобработки результатов.

Система реализует последовательность этапов обработки изображений бланков с использованием нейросетевых моделей трансформерного типа.

Интеграционная архитектура и поток данных представлены на рисунке 4.

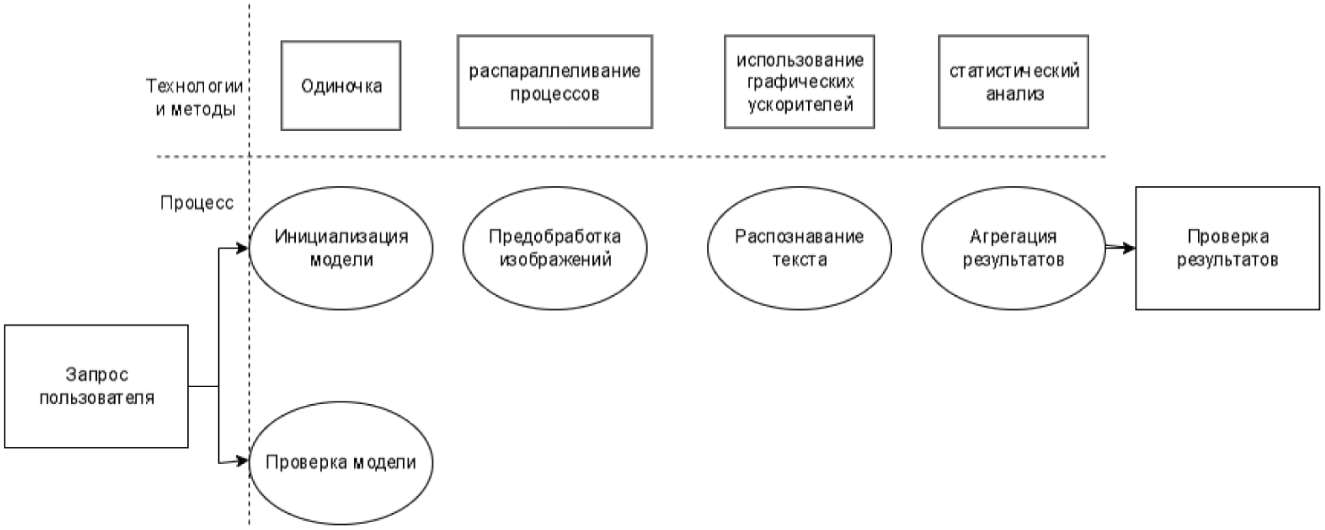

Рисунок 4. Интеграционная архитектура и поток данных

Архитектура следует паттерну pipeline с явным разделением ответственности:

– компонент инициализации модели – ModelInitializer;

– компонент предобработки изображений – ImagePreprocessor;

– компонент распознавания текста – TextRecognizer;

– компонент агрегации результатов – ResultAggregator.

Опишем компонент инициализации модели (ModelInitializer). Архитектурная инициализация нейросетевых компонентов осуществляется через загрузку предобученных весов трансформерной архитектуры VisionEncoderDecoder, специфически адаптированной для распознавания рукописного русского текста. Данный процесс включает не только загрузку параметров модели, но и оптимизацию вычислительного графа для режима «вывода», что подразумевает отключение механизмов регуляризации (отсев, пакетная нормализация в режиме обучения) и перевод модели в состояние оценки (eval()). Критически важным аспектом является автоматическая детекция доступных вычислительных ресурсов с приоритетом использования CUDA-ядер графических ускорителей NVIDIA или в случае техники Apple MPS (Metal Performance Shaders), что обеспечивает значительное ускорение обработки за счет параллельных матричных операций. Дополнительно применяется квантизация весов в формат FP16 для оптимизации использования памяти видеокарты без существенной потери точности, а также статическая компиляция вычислительного графа через torch.jit для исключения накладных расходов интерпретатора Python (листинг кода 1).

```
def load_pretrained_components(self)->tuple[TrOCRProcessor,VisionEncoderDecoderModel]:
        # Реализация с обработкой исключений
        try:
            processor = TrOCRProcessor.from_pretrained(
                "kazars24/trocr-base-handwritten-ru",
                cache_dir="./model_cache",
                local_files_only=False
            )
            model = VisionEncoderDecoderModel.from_pretrained(
                "kazars24/trocr-base-handwritten-ru",
                torch_dtype=torch.float16,  # Mixed precision
                cache_dir="./model_cache"
            )
        except ConnectionError:
            # Fallback на локальную версию
            processor, model = self._load_local_backup()
def configure_hardware(self, model: VisionEncoderDecoderModel) -> None:
        if hasattr(torch.cuda, 'empty_cache'):
            torch.cuda.empty_cache()
        # Оптимизация для inference
        model.to(self.device)
        model.eval()  # Отключение dropout, batch normalization в train mode
def optimize_inference(self, model: VisionEncoderDecoderModel) -> None:
        if self.device == "cuda":
            model.half()  # FP16 precision
        # Fusion для линейных слоев и активаций
        torch.backends.cudnn.benchmark = True # Auto-optimize CuDNN
```

Листинг кода 1. Организация компонента инициализации модели

Основа компонента базируется на принципах transfer learning, где предобученная на больших датасетах модель дообучается (fine-tuning) на специфической задаче распознавания рукописного текста, что позволяет достичь высокой точности даже при ограниченном количестве обучающих данных. Математически это выражается в оптимизации функции потерь на целевом датасете с замороженными нижними слоями энкодера, которые отвечают за выделение низкоуровневых признаков изображения и адаптацию верхних слоев под специфику домена.

Опишем компонент предобработки изображений (ImagePreprocessor). Ключевым этапом является применение алгоритма адаптивного выравнивания гистограммы с ограничением контрастности (Contrast Limited Adaptive Histogram Equalization, CLAHE) для улучшения контрастности текстовых областей, который работает путем разделения изображения на квадраты (тайлы M*N) и выравнивания гистограммы яркости в каждом тайле с ограничением усиления шума через ограничивающий параметр, *clipLimit=3·M·N/255* (листинг кода 2).

```
def apply_clahe(self, image: np.ndarray) -> np.ndarray:
        # Реализация с оптимизацией параметров
        if image.dtype != np.uint8:
            image = (image * 255).astype(np.uint8)
        clahe = cv2.createCLAHE(
            clipLimit=3.0,
            tileGridSize=(8, 8) # Оптимально для текста
        )
        return clahe.apply(image)
```

Листинг кода 2. Адаптивная бинаризация с CLAHE

Математически данный метод использует уравнение трансформации (1) и основан на преобразовании функции распределения интенсивности пикселей L с целью максимизации энтропии изображения, что особенно важно для обработки сканированных документов с неравномерным освещением или низким качеством печати.

$$T(i) = (L-1) \cdot \sum_{j=0}^{i} p(j), \quad (1)$$

где $p(j)=H(j)/(M*N)$, $H(i)$ – гистограмма интенсивности.

Алгоритмы компьютерного зрения в данном компоненте реализуют сложную последовательность обработки графических данных, начинающуюся с адаптивной сегментации изображения на регионы интереса (ROI) по заранее определенной координатной сетке, используя матрицу трансформации (2), параметры которой оптимизированы под специфический формат бланка ответов (листинг кода 3).

```
def extract_answer_cells(self, image: Image.Image, num_questions: int) -> list[Image.Image]:
        # Конфигурируемые параметры сетки
        grid_config = {
            'x_start': 300, 'y_start': 1033,
            'cell_width': 1200, 'cell_height': 90,
            'row_spacing': 15, 'col_spacing': 117,
            'questions_per_column': 20
        }
        pieces = []
        image_array = np.array(image)

        # Адаптивная коррекция координат через анализ границ
        corrected_coords = self._detect_grid_boundaries(image_array)
        for col in range((num_questions - 1) // grid_config['questions_per_column']
+ 1):
      for row in range(min(num_questions - col * grid_config['questions_per_column'],
                           grid_config['questions_per_column'])):
            # Расчет координат с коррекцией
            x, y = self._calculate_cell_position(col, row, corrected_coords)
            cell = image.crop((x, y, x + grid_config['cell_width'],
                               y + grid_config['cell_height']))
            pieces.append(cell)
        return pieces
```

Листинг кода 3. Геометрическое сегментирование ROI

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \quad (2)$$

Геометрическое преобразование включает не только простое вырезание регионов по координатам, но и адаптивную коррекцию положения границ через методы детекции линий, используя преобразование Хафа и морфологические операции эрозии и расширения для компенсации возможных смещений или поворотов бланка при сканировании. Для каждого выделенного региона дополнительно применяется нормализация размеров и разрешения, что обеспечивает согласованность входных данных для нейросетевого энкодера и повышает стабильность работы модели.

Опишем компонент распознавания текста (TextRecognizer).

Нейросетевая архитектура обработки текста построена на комбинации визуального трансформера (Vision Transformer, ViT) в качестве энкодера и авторегрессионного трансформера в качестве декодера, что представляет собой передовой подход для задач преобразования изображения в текст.

Энкодер преобразует изображение x в последовательность векторных представлений z через разбиение на N перекрывающихся патчей x_p^i размером 16x16 пикселей с последующей позиционной кодировкой и обработкой многослойным трансформером (3), способным улавливать глобальные контекстные зависимости между различными частями изображения (листинг кода 4).

```
def encode_image(self, pixel_values: torch.Tensor) -> torch.Tensor:
        # Forward pass через ViT энкодер
        with torch.no_grad():
            encoder_outputs = self.model.encoder(
                pixel_values=pixel_values,
                output_attentions=False,
                output_hidden_states=True
            )
        return encoder_outputs.last_hidden_state
```

Листинг кода 4. Энкодинг изображения

$$z_0 = \begin{bmatrix} x_{cl} & x_p^1 & \cdots & x_p^N \end{bmatrix} + E,$$
$$z'_l = MHSA(LN(z_{l-1})) + z_{l-1}, \quad (3)$$
$$z_l = MLP(LN(z'_l)) + z'_l,$$

где x_{cl} – классификационный токен, $MHSA$ – «многоголовое самовнимание» Multi-Head Self-Attention, LN – нормализация слоев Layer Normalisation, MLP – многослойный перцептрон (Multi-Layer Perceptron).

Декодер генерирует текстовую последовательность токен за токеном, используя механизм внимания как к предыдущим токенам, так и к энкодированному представлению изображения, что позволяет учитывать как лингвистический контекст, так и визуальные особенности рукописного текста.

Алгоритм поиска луча (beam search) применяется для поиска наиболее вероятной последовательности токенов с балансом между качеством и вычислительной эффективностью, используя параметр num_beams для контроля ширины поиска и length_penalty для предотвращения генерации слишком коротких или длинных последовательностей (листинг кода 5).

```
def encode_image(self, pixel_values: torch.Tensor) -> torch.Tensor:
        # Forward pass через ViT энкодер
        with torch.no_grad():
            encoder_outputs = self.model.encoder(
                pixel_values=pixel_values,
                output_attentions=False,
                output_hidden_states=True
            )

        return encoder_outputs.last_hidden_state
```

Листинг кода 5. Авторегрессивный декодинг текста

Для каждого сгенерированного токена вычисляется оценка уверенности через softmax-нормализацию наборов чисел (логитов) с последующим усреднением по всей последовательности, что дает вероятностную оценку достоверности распознавания на уровне всего ответа, а не отдельных символов (листинг кода 6).

```
def calculate_token_confidence(self, scores: list[torch.Tensor],
                               sequences: torch.Tensor) -> list[float]:
        confidences = []
        for i, (token_scores, token_id) in enumerate(zip(scores, sequences[0][1:])):
            # Softmax нормализация
            probabilities = F.softmax(token_scores[0], dim=-1)
            token_prob = probabilities[token_id.item()].item()
            confidences.append(token_prob)
        # Статистическая обработка
        return {
            'mean_confidence': np.mean(confidences),
            'min_confidence': np.min(confidences),
            'confidence_std': np.std(confidences),
            'token_confidences': confidences}
```

Листинг кода 6. Вычисление оценки уверенности

Опишем последний компонент агрегации результатов (ResultAggregator).

Статистический анализ качества распознавания включает многоуровневую систему валидации, основанную на комбинации метрик оценки уверенности, лингвистических правил и проверок согласованности формата ответа. Помимо простого порогового сравнения средней уверенности модели, применяется анализ распределения уверенности по токенам с вычислением стандартного отклонения и минимального значения, что позволяет выявлять случаи, когда модель не уверена в отдельных частях ответа даже при высокой средней оценке (листинг кода 7).

```
def validate_confidence(self, raw_results: list,
                        thresholds: dict = None) -> list[CheckerAnswer]:
        default_thresholds = {
            'high_confidence': 0.95,
            'medium_confidence': 0.85,
            'low_confidence': 0.7,
            'min_length': 1,
            'max_length': 50
        }
        validated_results = []
        for i, (text, confidence) in enumerate(raw_results):
            # Многофакторный анализ качества
            quality_score = self._compute_quality_score(text, confidence)
            error_flag = quality_score < default_thresholds['medium_confidence']
            validated_results.append(CheckerAnswer(
                user_identity=raw_results[0][0],  # Первый элемент - идентификатор
                result=text,
                question=i + 1,
                error=error_flag,
                accuracy=quality_score,
                confidence_metrics=self._get_detailed_metrics(quality_score)
            ))
        return validated_results
```

Листинг кода 7. Пороговая классификация результатов

Структурирование и сериализация результатов осуществляются в соответствии с заранее определенной онтологией данных, включающей идентификационные атрибуты работы, временные метки обработки и детализированную информацию по каждому вопросу с классификацией ошибок по типам и степени серьезности. Система логирования фиксирует не только конечные результаты, но и промежуточные метрики производительности, такие как пропускная способность, задержка и использование, что позволяет проводить последующий анализ и оптимизацию работы последовательности этапов (листинг кода 8).

```
def log_processing_metrics(self, start_time: float,
                           image_count: int) -> dict:
        processing_time = time.time() - start_time
        metrics = {
            'total_time': processing_time,
            'images_per_second': image_count / processing_time,
            'average_latency': processing_time / image_count,
            'timestamp': datetime.datetime.now().isoformat(),
            'hardware_utilization': self._get_hardware_stats()
        }
        # Структурированное логирование
        self.logger.info("Processing metrics", extra={'metrics': metrics})
        return metrics
```

Листинг кода 8. Определение временных метрик и логирование

Тестирование полученной системы проводилось с использованием бланков ЕГЭ на основе 250 образцов русского рукописного текста, состоящего из:

– 100 образцов – ячейки с аккуратным заполнением (категория А);

– 75 образцов – ячейки с текстом, содержащие небольшие наклоны букв (до 5 градусов вправо) или нечастые и небольшие выходы за границы заполненных ячеек (категория Б);

– 75 образцов – ячейки с текстом, содержащие сильный наклон букв (от 6 до 20 градусов вправо), пересекающиеся буквы, а также плохо пропечатанный бланк (категория В).

Был проведен сравнительный анализ с рассмотренными ранее системами по скорости, точности и уверенности распознавания (таблица 1).

Таблица 1. Сравнительные характеристики

Система	Кат. А	Кат. Б	Кат. В	Средняя V, в сек.	Общая точность	Средняя уверенность	Корреляция уверен.-точн.
Разработанная система	94,2 %	87,5 %	72,3 %	2,3	86,7 %	0,89	0,82
Tesseract 5.0	45,8 %	38,2 %	25,1 %	1,8	38,7 %	0,65	0,58
EasyOCR	68,9 %	62,4 %	48,7 %	1,5	61,4 %	0,78	0,71
PaddleOCR	82,3 %	75,6 %	58,9 %	3,1	73,2 %	0,84	0,76

Как и предполагалось, разработанная система более чувствительна к геометрическим искажениям текста и нечетко пропечатанным линиям ячейки. В результате анализа были выявлены 4 типичные проблемы распознавания:

– слипшиеся символы, в частности буквы «м», «ш», «щ» и выход текста за границы ячейки (~23 % случаев);

– путаница с парами цифр 1 и 7, 3 и 8, 0 и 6 (~18 % случаев);

– непропечатанные линии ячейки (~15 % случаев);

– ошибки определения границы ячеек в случае смещения при сканировании/печати бланка (~12 % случаев).

Подводя итог, можно сказать, что реальная производительность программного комплекса для формализованных бланков составляет от 85 до 89 % точности при адекватных условиях сканирования и заполнения.

ЗАКЛЮЧЕНИЕ И ВЫВОДЫ

В ходе проектирования была сформирована архитектура комплекса, включающая модули предобработки изображений с функциями выравнивания, шумоподавления и сегментации, нейросетевого распознавания рукописного текста, автоматической проверки ответов и удобный веб-интерфейс для загрузки работ и просмотра результатов, что в совокупности обеспечивает единое и интуитивно понятное решение для преподавателя.

Результаты распознавания агрегируются в структуру данных, содержащую для каждого вопроса распознанный ответ, метрику уверенности и флаг, указывающий на возможную ошибку.

Разработанный программный комплекс демонстрирует хорошие результаты по сравнению с аналогами, но требует соблюдения некоторых условий эксплуатации. Комплекс запускается и стабильно работает на локальном сервере и готов к внедрению в образовательный процесс.

СПИСОК ЛИТЕРАТУРЫ

1. Кому нужно программное обеспечение как услуга // Хабр. – URL: https://habr.com/ru/companies/first/articles/695036/ (дата обращения: 12.09.2025).
2. FastAPI documentation. – URL: https://fastapi.tiangolo.com (date of application: 12.09.2025).
3. JaidedAI/EasyOCR: Ready-to-use OCR // GitHub. – URL: https://github.com/JaidedAI/EasyOCR (date of application: 12.09.2025).
4. OCR vs. ICR: Document processing tech compared // Astera. – URL: https://www.astera.com/type/blog/ocr-vs-icr-all-the-differences/ (date of application: 12.09.2025).
5. PaddleOCR Documentation // PaddleOCR. – URL: https://www.paddleocr.ai/main/en/index.html (date of application: 12.09.2025).
6. Pydantic documentation // Pydantic Contributors. – URL: https://pydantic-docs.helpmanual.io (date of application: 12.09.2025).
7. Smith, R. An overview of the Tesseract OCR engine / R. Smith // Proc. Ninth International Conference on

Document Analysis and Recognition (ICDAR-2007). – Curitiba, Brazil, 2007. – P. 629–633.

8. SQLAlchemy documentation // SQLAlchemy Authors. – URL: https://docs.sqlalchemy.org (date of application: 12.09.2025).
9. Yandex Vision OCR documentation // Yandex.Cloud. – URL: https://cloud.yandex.ru/docs/vision/ocr (date of application: 12.09.2025).