Санкт–Петербургский государственный университет

ИГНАТЬЕВ Денис Игоревич

Выпускная квалификационная работа

Создание системы для автоматизации сбора и обработки данных для машинного обучения

Уровень образования: бакалавриат
Направление 02.03.02 «Фундаментальная информатика и информационные технологии»
Основная образовательная программа СВ.5003.2017 «Программирование и информационные технологии»
Профиль «Автоматизация научных исследований»

Научный руководитель:
доцент, кафедра компьютерных технологий и систем, к.ф.-м.н. Погожев С. В.

Рецензент:
доцент, кафедра технологии программирования, к.т.н. Блеканов И. С.

Санкт-Петербург
2021 г.

Содержание

Введение

В последние годы машинное обучение нашло свое применения во многих отраслях человеческой деятельности, в том числе и в научных исследованиях. Обучение с учителем, как частный случай машинного обучения, является одним из популярных подходов для решения множества практических и исследовательских задач. Однако успех применения алгоритмов обучения с учителем зависит от количества и качества имеющихся данных. Данные, необходимые для обучения с учителем, должны быть размечены, то есть каждый элемент данных должен быть снабжен некоторой меткой, которую впоследствии алгоритм будет предсказывать. Отсутствие размеченных данных, их малый объем или низкое качество могут стать серьезной проблемой для использования алгоритмов машинного обучения.

Сбор и накопление размеченных данных зачастую является узким местом в научных исследованиях или же в практических задачах, связанных с применением машинного обучения. Как правило, процесс разметки данных требует кропотливой ручной работы множества экспертов, при котором возникают проблема человеческого фактора – принятие человеком неверных решений – и, как следствие, проблема снижения качества получаемых в результате работы экспертов данных. Стоит добавить, что процесс ручной разметки является крайне трудоемким, а объемы данных, которые необходимо разметить могут достигать сотни тысяч элементов. По указанным выше причинам, многие исследовательские группы заинтересованы в оптимизации процесса разметки данных.

Можно выделить два основных направления оптимизации процесса ручной разметки данных – это ускорение процесса и повышение качества разметки получаемых в результате работы экспертов данных. На текущий момент существует множество программных инструментов, решающих в той или иной степени задачи ускорении разметки и повышения качества собираемых данных. О популярности и важности этих задач также говорит большое количество современных публикаций и множество разрабатываемых методов, связанных с качественным улучшением процесса разметки данных.

В рамках научных исследований, к инструментам разметки данных могут

предъявляться следующие ключевые требования:

- Инструмент должен распространяться под открытой лицензией.
- Приложение может быть легко установлено на локальный компьютер пользователя, либо же развернуто в локальной компьютерной сети организации.
- Приложение предлагает большой набор инструментов для выполнения различных задач по разметке данных.
- Приложение предполагает многопользовательский режим работы.
- В приложении используются различные современные методы для ускорения процесса разметки данных.
- Архитектура приложение позволяет легко масштабировать инструмент под решение новых задач.
- Приложение может горизонтально масштабироваться при увеличении количества пользователей или объема размечаемых данных.
- Приложение осуществляет контроль целостности и качества данных.
- Отказоустойчивость приложения.

На данный момент среди инструментов разметки данных отсутствуют те, которые в полной мере удовлетворяют всем перечисленным ключевым требованиям.

Целью данной работы является проектирование архитектуры инструмента для автоматизации процесса ручной разметки данных, отвечающего указанным ключевым требованиям. Сферой применения данного инструмента предполагается сфера научных исследований, связанных с применением машинного обучения.

Список терминов

- ORM (Object-Relational Mapping) – техника преобразования данных между несовместимыми системами типов с использованием объектно-ориентированных языков программирования.
- Краудсорсинг – привлечение для решения какой-либо задачи широко круга лиц на добровольных началах с использованием информационных технологий.
- Микросервисная архитектура – подход к разработке программного обеспечения, при котором приложение разбивается на небольшие автономные компоненты (микросервисы) с четко определенными интерфейсами.
- Модульное тестирование – процесс проверки отдельных модулей программы на корректность работы.
- Сериализация – преобразование объекта в определенный формат (например, в последовательность байтов), из которого объект может быть восстановлен обратно.
- СУБД – система управления базами данных.
- Технологический стек – набор инструментов, языков программирования и фреймворков, применяющихся при реализации проекта.
- Требования к ПО – совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации.
- Утилита – вспомогательная компьютерная программа в составе общего программного обеспечения для выполнения специализированных типовых задач.
- Фреймворк – программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Постановка задачи

Для достижения цели работы были поставлены следующие задачи:

1. Провести исследование существующих программных инструментов для создания размеченных наборов данных.
2. Изучить описанные в научной литературе современные подходы для ускорения процесса разметки данных и повышения качества формируемых наборов данных.
3. Сформировать требования к разрабатываемой системе.
4. Разработать программную архитектуру приложения.
5. Разработать программный инструмент для разметки наборов данных с учетом сформированных требований и программной архитектуры.
6. Произвести теоретическую оценку эффективности работы системы.
7. Произвести тестирование разработанного приложения.

Обзор литературы

В современной научной литературе активно рассматривается задача создания размеченных наборов данных как со стороны проектирования систем для разметки данных, так и со стороны создания новых подходов, позволяющих повысить эффективность подобных систем.

В статье [1] описываются общие принципы создания систем для разметки данных с использованием краудсорсинга. Авторы статьи рассматривают ключевые компоненты системы, которые позволяют организовать эффективную систему ручной разметки данных с использованием краудсорсинга: декомпозиция задач разметки, простая и понятная инструкция к заданиям, легкий в использовании интерфейс, техники контроля качества разметки, методы агрегации пользовательской разметки, вознаграждения. Авторы статьи приводят описание того, как может быть организован процесс эффективного создания некоторых типов разметки. В основе данного процесса лежит принцип конвейера: представление сложного задания в виде набора простых задач, которые могут быть быстро решены исполнителями. В настоящей работе идея разбиения сложных задач разметки на простые была использована для формирования набора базовых типов разметки, которые являются достаточно простыми для того, чтобы пользователи могли их достаточно быстро выполнять.

В научной литературе представлен ряд работ, описывающих способы ускорения процесса ручной разметки за счет минимизации количества действия, которые необходимо произвести пользователю, чтобы разметить один элемент из набора данных. Метод выделения объекта на фрагменте изображения, получивший название GrabCut, описан в статье [2]. Данный метод позволяет отделять автоматически или полуавтоматически объект на изображении от фона, что может быть применено для ускорения сегментации изображений. Для выделения объекта пользователь должен выделить объект прямоугольником, после чего метод автоматически выделит маску объекта. Инструмент для сегментации изображений ByLabel [3] автоматически определяет потенциальные границы на изображении для того, чтобы пользователь самостоятельно выбрал нужные границы для формирования контура маски объекта. Предложенный авторами статьи подход сокращает количество дей-

ствий, которые должен произвести пользователь для выделения маски объекта на изображении. Другой инструмент Polygon-RNN [4] также автоматизирует процесс сегментации изображений за счет сокращения необходимого количества действия для выделения одного объекта, в лучшем случае, до двух кликов: пользователю необходимо выделить объект прямоугольником (это можно сделать всего за два клика), а нейронная сеть специальной архитектуры предсказывает контур объекта в виде замкнутой ломанной. Особенностью инструмента является то, что пользователь может исправлять ошибки модели, корректируя контур объекта вручную.

Другим популярным методом для ускорения процесса разметки данных является использование алгоритмов машинного обучения для автоматизации ручной работы экспертов. В статье [5] авторами предлагается идея полуавтоматической разметки с использованием алгоритмов машинного обучения, заключающаяся в том, что при накоплении достаточно объема размеченных экспертами данных, происходит обучение алгоритма на собранных данных, который осуществляет разметку оставшегося набора данных. После обучения модели экспертам ставится задача оценки правильности полуавтоматической разметки. Ускорение процесса разметки в данном случае заключается в том, что ручная проверка правильности разметки является зачастую более простой задачей, чем непосредственно разметка. Идея полуавтоматической разметки, предложенная авторами статьи, легла в основу функционирования системы автоматической разметки в разрабатываемом приложении.

Эффективность применения машинного обучения для автоматизации процесса разметки данных напрямую зависит от необходимого для обучения модели количества данных, скорости обучения моделей и точности их работы. Метод активного обучения призван существенно снизить количество данных, необходимых для обучения модели [6]. Данный метод заключается в итеративном обучении модели на некотором подмножестве объектов и отборе объектов для включения их в обучающую выборку. Отбор объектов осуществляется таким образом, чтобы одновременно сократить размер обучающей выборки и добиться хорошей точности алгоритма. Метод активного обучения хорошо применим в системах разметки данных, так как он предполагает то, что часть исходных данных может быть не размечена и реализует отбор тех элементов,

которые экспертам необходимо разметить в первую очередь.

Метод активного обучения объединяет большое количество различных реализаций алгоритмов для отбора наиболее «интересных» для разметки элементов данных. Данные алгоритмы зачастую индивидуальны для различных моделей машинного обучения. В статье [7] приводится классификация методов активного обучения на две группы – для классических моделей и для моделей глубокого обучения. Методы активного обучения для классических алгоритмов машинного обучения хорошо изучены и описаны в статье [8]. Описание того, как активное обучение может быть применено для моделей глубокого обучения, приведено в статье [9]. Помимо этого, авторы статьи предлагают алгоритм активного обучения для алгоритмов глубокого обучения для детектирования объектов на изображении (таких, как модели семейства YOLO [10, 11, 12, 13] или SSD [14]), который может быть использован для построения системы полуавтоматической разметки изображений.

В системах ручной разметки зачастую один и тот же элемент данных размечается сразу несколькими экспертами для повышения точности разметки [6]. При использовании данного подхода возникает задача агрегации результатов разметки для получения итоговой разметки.

Обзор методов агрегации ручной разметки приводится в статьях [6, 7]. Однако многие из описанных в литературе методов применимы лишь для задач классификации объектов. Идея метода для агрегации результатов выделения исполнителями ограничивающих рамок объектов на изображении (данный тип разметки широко распространен в сфере машинного зрения) с использованием алгоритма DBSCAN приведена в статье [15].

В результате проведенных исследований литературных источников, были выделены основные подходы для повышения эффективности ручной разметки:

1. Использование методов для уменьшения количества действий, которые должен выполнить пользователь для разметки одного элемента данных.

2. Использование алгоритмов машинного обучения для автоматизации процесса разметки.

3. Использование различных алгоритмов агрегации результатов разметки

для контроля качества формируемых наборов данных.

4. Использование активного обучения для выбора тех объектов из набора, разметка которых позволит добиться лучшей точности работы алгоритма машинного обучения.

5. Использование различных методов для назначения заданий пользователям.

6. Декомпозиция сложной задачи разметки на более простые подзадачи для ускорения процесса разметки.

Глава 1. Обзор инструментов для разметки данных

В данном разделе рассмотрены наиболее популярные программные решения для разметки данных, их основные возможности и ограничения.

1.1 Amazon Mechanical Turk

Amazon Mechanical Turk (MTurk) [1] – это краудсорсинговая веб-платформа для координированного выполнения различных задач, которые называются «HIT» (Human Intelligence Tasks), с использованием человеческих ресурсов. Охват возможных задач, которые могут быть решены с использованием MTurk, широк и, в частности, включает в себя задачи ручной разметки наборов данных.

Пользователи MTurk делятся на два типа: Заявители (Requesters) и Сотрудники (Providers). Заявители размещают задания на платформе, а Сотрудники выполняют их. Вознаграждением Сотрудников за выполнение заданий является денежная выплата в размере, установленном Заявителем [16].

1.2 Yandex Toloka

Сервис Yandex Toloka[2] является краудсорсинговым проектом для быстрой разметки больших объемов данных [17]. Данный сервис использует модель постановки и выполнения задач, аналогичную MTurk. Выполнение задач также производится на коммерческой основе.

1.3 LabelMe

На рынке инструментов для разметки данных существует ряд открытых платформ для коллективной разметки данных. Открытые платформы разметки данных зачастую используются в небольших проектах, а также в сфере научных исследований. Наиболее известным представителем подобных платформ является веб-сервис для коллективной разметки изображений LabelMe[3]. Данный сервис предоставляет возможность размечать объекты на изображениях при помощи полигонов [5]. К достоинствам данного сервиса можно

[1] https://www.mturk.com/

[2] https://toloka.yandex.ru/

[3] http://labelme.csail.mit.edu/Release3.0/

отнести простоту использования, к недостаткам – отсутствие приватности загружаемых в систему данных. Инструмент LabelMe также поддерживает локальную установку, что может быть полезно для работы с конфиденциальными данными.

1.4 CVAT

Популярным инструментом для разметки данных в области компьютерного зрения является веб-приложение CVAT[4]. CVAT предоставляет широкий набор инструментов для создания размеченных наборов данных. Основные возможности инструмента рассмотрены в статье [18]. В качестве особенностей данного инструмента можно выделить наличие различных средств для ускорения процесса разметки данных, такие как, например, автоматическое выделение контуров объектов по ключевым точкам, а также возможность автоматически размечать данные с использованием обученных моделей.

1.5 Amazon SageMaker Ground Truth

Amazon SageMaker Ground Truth[5] – сервис, основной задачей которого является ускорение процесса разметки больших наборов данных. Данный сервис предоставляет возможности как для ручной разметки данных, так и для автоматизированной.

Amazon SageMaker Ground Truth предоставляет возможность использовать автоматизированную разметку изображений с использованием активного обучения на больших наборах данных [19]. Сервис реализует следующий алгоритм:

1. Часть набора данных размечается исполнителями для формирования обучающей и тестовой выборки.

2. На сформированном наборе данных обучается модель. Предполагается, что модель для каждого поданного на вход объекта возвращает метки с указанием степени уверенности.

[4]https://github.com/openvinotoolkit/cvat

[5]https://aws.amazon.com/ru/sagemaker/groundtruth/

3. Модель используется для разметки тестовой выборки; определяется порог степени уверенности, дающий наилучшую точность предсказания.

4. Модель размечает оставшийся набор данных и те метки, на которых степень уверенности превосходит порог, принимаются как итоговая разметка данных.

5. Объекты, на которых модель продемонстрировала низкую степень уверенности, отправляются на ручную разметку.

6. Далее полученный набор данных, размеченный людьми, используется для повторного обучения модели. Данный алгоритм повторяется до тех пор, пока в наборе данных присутствуют неразмеченные элементы.

Глава 2. Сбор требований

Согласно книге [20] основными требованиями к ПО являются бизнес-требования (высокоуровневые цели), требования пользователей, функциональные требования и нефункциональные требования. В данном разделе описаны основные требования, предъявляемые к разрабатываемому инструменту для разметки данных.

2.1 Бизнес-требования

Бизнес-требования к разрабатываемому инструменту разметки вытекают из цели данной работы и могут быть сформулированы следующим образом: должен быть разработан инструмент для автоматизации процесса разметки данных для машинного обучения в сфере научных исследований. В основе процесса автоматизации может быть использован процесс, описанный в [5], и заключающийся в обучении модели на размеченных вручную данных и переключении пользователей с ручной разметки на проверку результатов работы модели.

2.2 Требования пользователей

Основные требования пользователей были сформированы на основе исследования существующих решений для разметки данных. Прежде всего было выделено три категории пользователей:

1. Категория «заказчики» – лица, желающие получить набор размеченных данных.

2. Категория «эксперты» – лица, осуществляющие разметку данных.

3. Администратор – лицо, осуществляющее техническое обеспечение системы.

Далее приведено описание основных требований пользователей разрабатываемого инструмента разметки данных:

1. Общие требования пользователей:

1.1. Администратор системы должен иметь возможность создавать новых пользователей.

1.2. Пользователи должны иметь возможность осуществлять вход в систему с использованием логина и пароля, выданного администратором системы.

2. Заказчик должен иметь возможность:

2.1. Загружать данные для разметки.

2.2. Просматривать данные, которые он когда-либо загружал в систему.

2.3. Создавать задания разметки данных.

2.4. Назначать экспертам задания.

2.5. Просматривать созданные им задания разметки данных.

2.6. Выгружать из системы размеченные данные.

3. Эксперт должен иметь возможность:

3.1. Просматривать назначенные для выполнения задания разметки.

3.2. При выполнении конкретного задания получать элемент данных для разметки, осуществлять разметку и отправлять результат.

3.3. Переключаться между заданиями ручной разметки и заданиями проверки результатов автоматической разметки.

2.3 Функциональные требования

Функциональные требования были сформированы, прежде всего, исходя из бизнес-требований и пользовательских требований.

1. Администрирование системы:

1.1. Администратор системы должен иметь возможность создавать новых пользователей, указывая их логин, пароль и роль.

2. Управление наборами данных:

2.1. Заказчик должен иметь возможность загружать наборы данных, указывая локальный путь к директории на своем персональном компьютере; присваивать имя загружаемому набору данных.

2.2. Заказчик должен иметь возможность просматривать загруженные наборы данных в виде списка с указанием названий и даты их загрузки.

3. Управление заданиями разметки:

3.1. Заказчик должен иметь возможность для заданного набора данных создавать задание разметки определенного типа; присваивать имя создаваемому заданию разметки и указывать параметры задания разметки.

3.2. Заказчик должен иметь возможность для выбранного задания разметки указывать экспертов, имеющих доступ к осуществлению разметки в рамках данного задания.

3.3. Заказчик должен иметь возможность выгружать разметку пользователей на любом этапе выполнения задания.

4. Разметка наборов данных:

4.1. Эксперт должен иметь возможность просматривать доступные для выполнения задания разметки с указанием их названий.

4.2. Эксперт должен иметь возможность переключаться между режимом ручной разметки и режимом проверки результатов автоматической разметки.

4.3. При выполнении конкретного задания эксперт должен иметь возможность получить элемент данных для разметки, осуществить разметку и отправить результат.

5. Автоматизация процесса разметки:

5.1. Система должна по мере накопления размеченных данных запускать процесс обучения моделей машинного обучения.

5.2. Система должна использовать обученные модели для осуществления автоматической разметки данных.

5.3. Система должна отправлять автоматически размеченные задания на ручную проверку пользователями.

5.4. Система должна сохранять одобренные пользователями результаты автоматической разметки и использовать их для последующего обучения моделей.

2.4 Нефункциональные требования

В данном разделе описаны нефункциональные требования к приложению и атрибуты качества разрабатываемого приложения.

1. Инструмент должен распространяться под открытой лицензией.
2. Инструмент должен быть прост в использовании, как со стороны заказчиков, так и со стороны экспертов, иметь минималистичный интерфейс.
3. Приложение должно иметь простой сценарий установки на локальный компьютер пользователя и в локальную компьютерную сеть организации.
4. Приложение должно обеспечивать многопользовательский режим работы.
5. Архитектура приложения должна позволять добавлять новые методы повышения эффективности разметки, новые алгоритмы функционирования компонентов системы.
6. Архитектура приложение позволяет легко масштабировать инструмент под решение новых задач.
7. Архитектура приложения должна быть потенциально масштабируемой при увеличении количества пользователей или объема размечаемых данных.
8. Приложение должно осуществлять контроль целостности и качества данных.

9. Приложение должно быть отказоустойчиво относительно сбоев в работе вспомогательных компонентов системы; приложение должно обеспечивать простоту восстановления работы системы после возникновения сбоев.

Глава 3. Программная архитектура приложения

В данном разделе приводятся общий концепт архитектуры приложения и аргументация его выбора, список компонентов системы.

3.1 Общее описание архитектуры

В результате анализа собранных требований, были выделены основные логические подсистемы проектируемого приложения.

Подсистема ручной разметки данных

К данной подсистеме относится вся функциональность, связанная с загрузкой данных в систему и их хранением, а также ручной разметкой данных. Стоит отметить, что данная подсистема приложения должна представлять собой автономно работающий сервис, который может использоваться без остальных подсистем приложения.

Подсистема автоматической разметки данных

Данная подсистема должна предоставлять функциональность по автоматизации разметки данных на основе сформированных экспертами наборов размеченных данных. Подсистема автоматической разметки потенциально может выполнять трудоемкие вычисления, поэтому обособление данной подсистемы на физическом уровне от других позволит повысить пропускную способность всего приложения.

Графический интерфейс приложения

Данная подсистема приложения должна предоставлять пользователям способ взаимодействия с приложением. Графический интерфейс приложения должен абстрагировать пользователей от способа функционирования подсистем приложения и предоставлять инструментарий для разметки данных.

Для технической реализации приложения было решено использовать микросервисную архитектуру. Приведем следующие аргументы в пользу выбора микросервисной архитектуры в данной работе:

- Приложение состоит из нескольких независимых компонентов. Использование микросервисов в данном случае позволит проще физически отделить компоненты друг от друга и обеспечить большую устойчивость приложения к сбоям в отдельных компонентах.

- Микросервисы, осуществляющие трудоемкие вычисления, могут быть физически отделены от всего приложения. Также использование микросервисной архитектуры может помочь горизонтально масштабировать систему при нехватке вычислительных ресурсов.

- Различные компоненты приложения могут быть более эффективно реализованы с использованием различных языков программирования (данный вопрос подробнее обсуждается в разделе 4.2).

3.2 Компоненты приложения

Первым делом рассмотрим микросервисы, формирующие подсистему ручной разметки данных. Графическое представление архитектуры подсистемы приведено на рисунке 1. Сплошные стрелки на диаграмме обозначают отношение «данный компонент использует другой компонент». Начало сплошных стрелок соответствует зависимым компонентам. Пунктирные стрелки обозначают сообщения, получаемые компонентом извне. Важно отметить то, что способ обмена сообщениями в архитектуре не описывается, так как является деталью реализации. Например, в различных реализациях сообщение может передаваться от одного компоненту другому напрямую, а может – через посредника.

MarkupDatabase

Реляционная база данных *MarkupDatabase* хранит основную информацию о пользователях инструмента разметки данных, исходные и размеченные данные. ER-модель базы данных приведена на рисунке 2.

Сущность *User* представляет собой пользователя системы. Каждый пользователь может иметь роли (сущность *Role*): *Customer* (Заказчик) и *Expert*

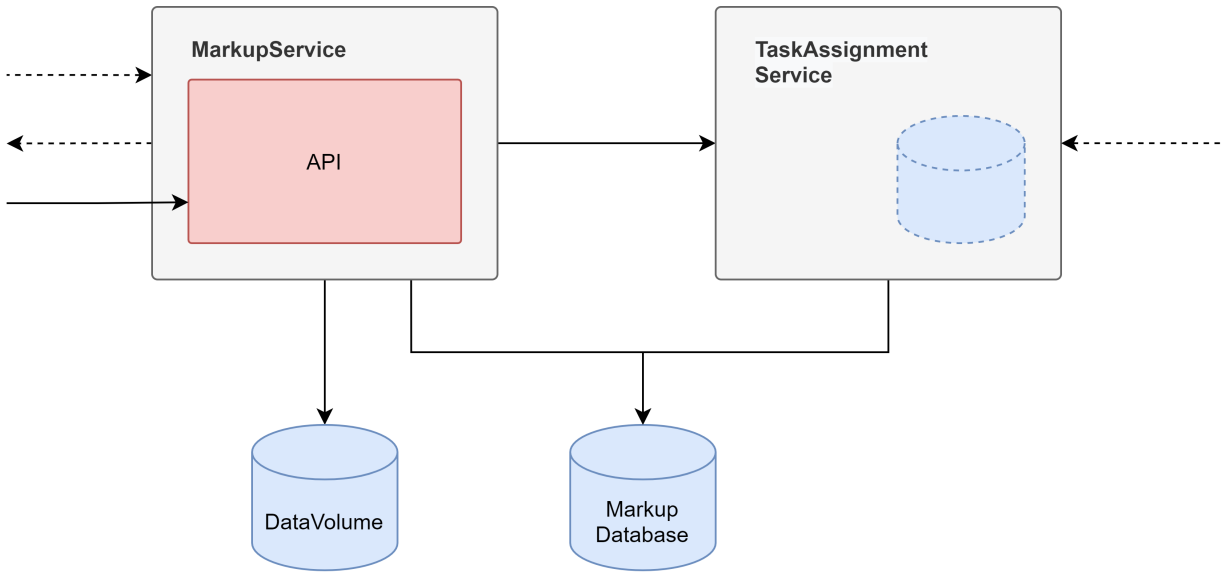


Рис. 1: Архитектура подсистемы ручной разметки данных.

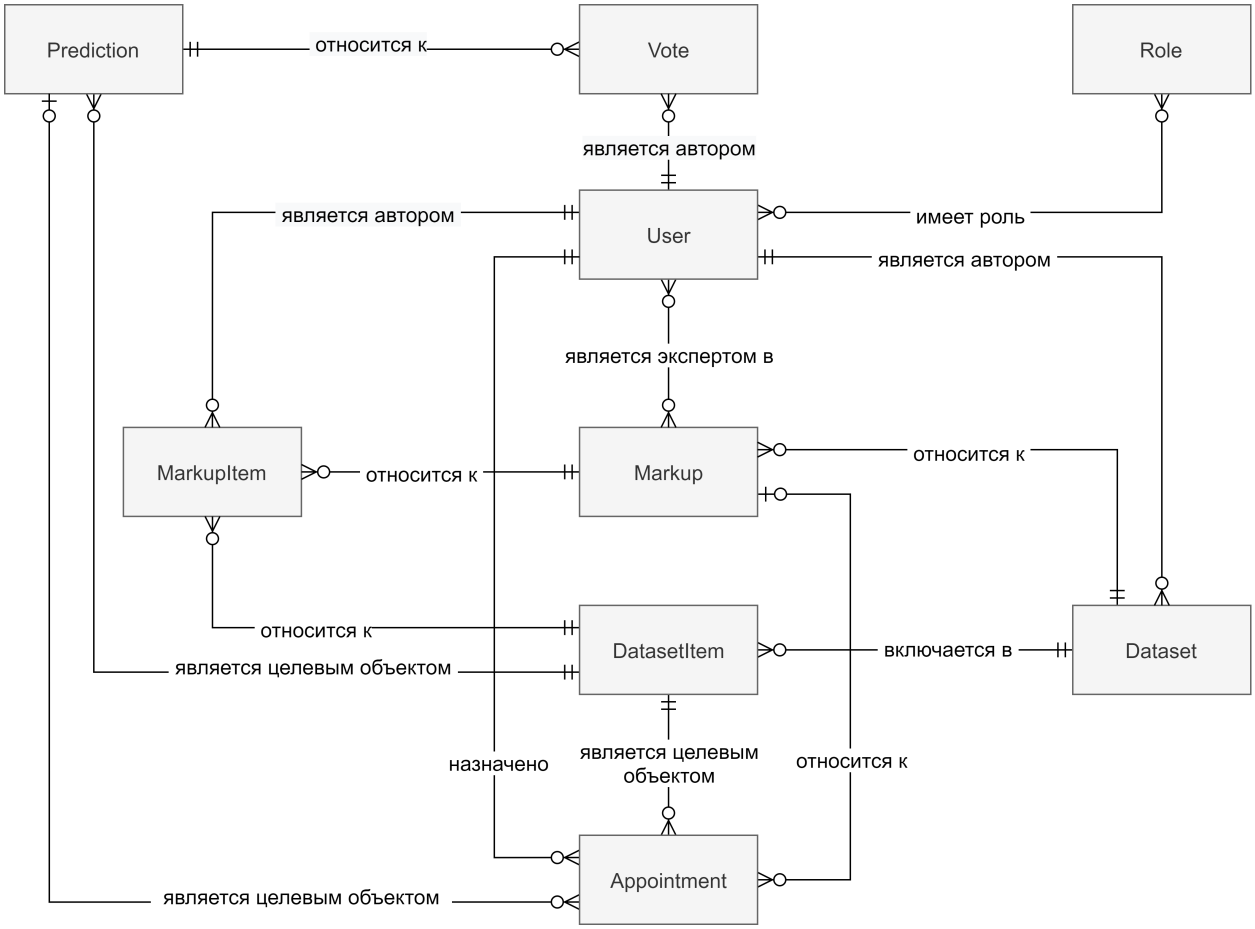


Рис. 2: ER-модель базы данных *MarkupDatabase*.

(Эксперт). Наборы данных, которые загружают для разметки заказчики, представлены сущностью *Dataset*. *Dataset* имеет автора, а также включает в себя набор элементов одной природы (изображения, тексты и пр.). Элемент набора данных представлен сущностью *DatasetItem*. Следует отметить, что *DatasetItem* не включает в себя сам элемент данных, а содержит только метаданные об элементе и способ доступа к нему, например некоторую ссылку.

Задание разметки, которое создают заказчики для формирования размеченного набора данных, представлено сущностью *Markup*. Каждое задание разметки может выполняться определенной заказчиком группой экспертов. Эксперты осуществляют разметку конкретных элементов данных из заданного набора. Результат разметки одного элемента данных одним экспертом в рамках задания разметки представлен сущностью *MarkupItem*. Непосредственно разметка не имеет четко заданной структуры, поэтому она должна храниться в реляционной базе в некотором сериализованном формате, например, в качестве строки в формате JSON. Данный подход является обоснованным, так как делают структуру базы независимой от типов разметки, которые позволяет осуществлять инструмент.

Автоматически сгенерированная разметка наборов данных представлена сущностью *Prediction*. Результаты проверки корректности автоматической разметки представлены сущностью *Vote*. *Vote* представляет собой ответ эксперта («да»/«нет») на вопрос «верна ли разметка для указанного элемента данных?». Сущность *Appointment* представляет собой назначение определенного задания конкретному эксперту.

DataVolume

DataVolume представляет собой директорию в файловой системе, в которой хранятся элементы загружаемых в систему наборов данных. Сущность *DatasetItem* базы данных *MarkupDatabase* ссылается на некоторый элемент данных, расположенный в *DataVolume*. Стоит отметить, что, хотя *DataVolume* отнесен к подсистеме ручной разметки, к нему имеют доступ микросервисы из других подсистем.

MarkupService

Микросервис *MarkupService* представляет собой обертку над *Markup-Database* и *DataVolume*. С данным микросервисом взаимодействует клиентская часть приложения, а также некоторые микросервисы из подсистемы автоматической разметки для получения необходимых данных.

TaskAssignmentService

Данный микросервис является вспомогательным для *MarkupService*. Основной задачей сервиса является назначение заданий экспертам. Архитектура *TaskAssignmentService* предполагает реализацию различных алгоритмов назначения заданий, которые, в общем случае, могут требовать хранения данных из различных источников (например, приоритизацию элементов данных с использованием активного обучения). Поэтому в архитектуре *TaskAssignmentService* предполагается наличие некоторого внутреннего хранилища.

Далее рассмотрим архитектуру подсистему автоматической разметки. Графическое представление архитектуры приведено на рисунке 3.

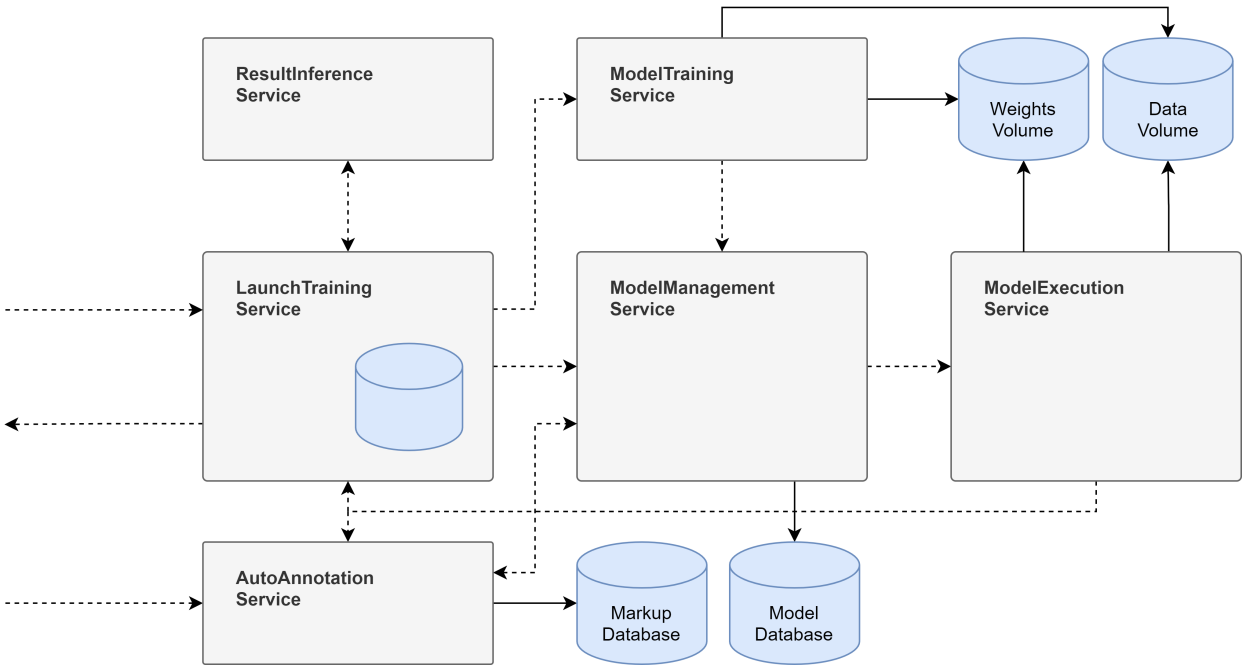


Рис. 3: Архитектура подсистемы автоматической разметки данных.

WeightsVolume

Данное хранилище представляет собой директорию в файловой системе, внутри которой в виде файлов располагаются веса моделей, обученных в процессе работы системы автоматической разметки.

ModelDatabase

База данных *ModelDatabase* содержит информацию о различных моделях машинного обучения, которые были обучены в процессе работы системы. Запись о каждой модели включает в себя тип задачи, которую решает модель, идентификатор задания разметки и путь к весам модели, располагающимся в *WeightsVolume*.

ModelExecutionService

Данный микросервис осуществляет запуск обученных моделей на переданном наборе данных. Для запуска модели на вход *ModelExecutionService* требуется подать набор данных, которые требуется разметить, путь к весам модели, а также всю необходимую информацию для чтения файла с весами (тип модели, тип задачи и пр.).

ModelManagementService

Микросервис *ModelManagementService* является оберткой над *ModelDatabase*. Задачами данного микросервиса являются:

- обработка событий запуска обучения моделей;
- обработка событий завершения обучения моделей;
- перенаправление запросов на автоматическую разметку данных *ModelExecutionService*.

ResultInferenceService

ResultInferenceService решает задачу агрегации накопленной разметки. Разметка, сформированная экспертами, а также разметка, которая генерирует-

ся автоматически, должны быть преобразованы в некоторую итоговую разметку. Итоговая разметка в дальнейшем используется для обучения моделей, либо возвращается заказчику как результат работы системы. Данный микросервис представляет собой некоторую библиотеку функций для агрегации различных типов разметки, которую могут использовать другие микросервисы.

ModelTrainingService

Данный сервис осуществляет обучение моделей на основе принимаемых на вход размеченных данных. В случае успешного обучения модели, ее веса сохраняются в *WeightsVolume*, а метаинформация о полученной модели отправляется в *ModelManagementService*.

TrainingLaunchService

Основной задачей *TrainingLaunchService* является запуск процесса обучения моделей. Прежде всего сервис определяет нужно ли начать обучение модели в данный момент времени. Действительно, так как обучение моделей достаточно ресурсоемкая задача, запускать обучение каждый раз, когда обновляются данные, было бы нерационально. При этом также стоит учитывать, что перед первым запуском обучения модели требуется накопить некоторый объем размеченных данных.

Следующей задачей, которую решает *TrainingLaunchService*, является подготовка данных для обучения моделей. Данный сервис собирает разметку данных, агрегирует с использованием *ResultInferenceService* и отправляет предобработанные данные в *ModelTrainingService*.

AutoAnnotationService

Данный микросервис осуществляет автоматическую разметку данных в определенные моменты времени и сохраняет ее в *MarkupDatabase*. Основной проблемой, которую решает сервис, является определение того, стоит ли продолжать производить автоматическую разметку данных или же обученная модель показывает низкое качество разметки и ее использование нерационально. Другой проблемой, которую решает *AutoAnnotationService*, является

поддержание в базе данных необходимого количества автоматически размеченных данных для непрерывной работы экспертов во время валидации разметки.

Клиентская часть приложения

Клиентская часть приложения (*Frontend*), представляющая собой графический интерфейс инструмента разметки, должна взаимодействовать с серверной частью приложения посредством некоторого фиксированного API. Способ взаимодействия клиентской и серверной частей приложения приведен на рисунке 4.

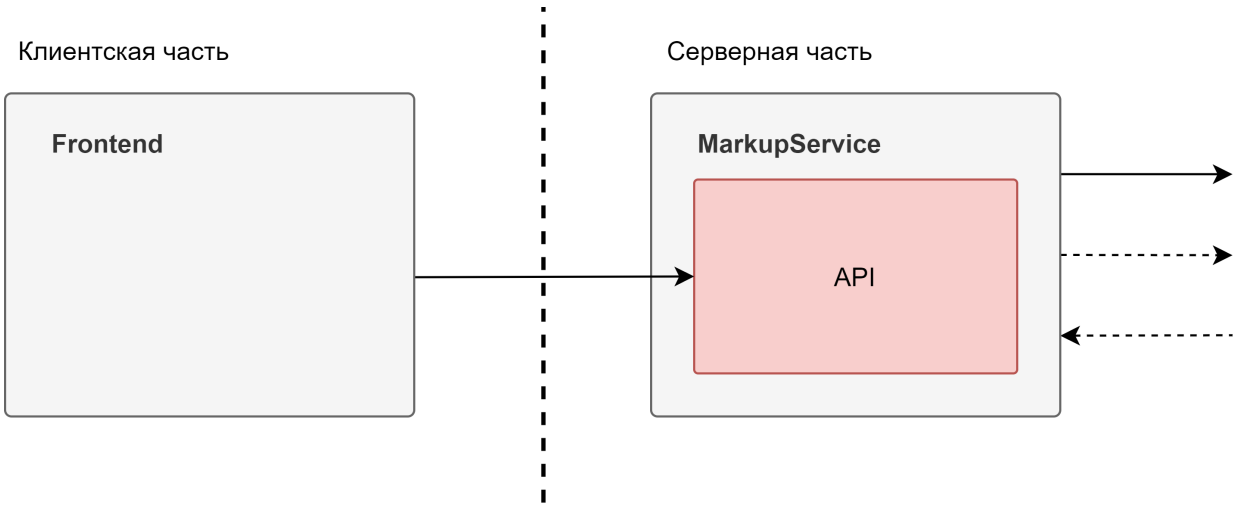


Рис. 4: Способ взаимодействия клиентской и серверной частей приложения.

3.3 Сценарии взаимодействия микросервисов

При проектировании системы, состоящей из большого количества компонентов, следует описать основные сценарии взаимодействия компонентов. При проектировании архитектуры приложения были выделены следующие сценарии:

1. Сценарий ручной разметки. Целью данного сценария является накопление данных, размеченных экспертами.

2. Сценарий обучения модели. Цель сценария состоит в обучении модели с использованием собранных данных.

3. Сценарий автоматической разметки. Целью данного сценария является формирование набора автоматически размеченных данных для последующей отправки экспертам для оценки их корректности.

Далее подробно рассмотрим последовательность взаимодействия пользователей с системой и компонентов внутри нее для достижения цели каждого сценария. Для графического представления каждого сценария будем использовать диаграмму последовательности (Sequence Diagram) [21], позволяющую наглядно описать взаимодействие компонентов системы с течением времени.

3.3.1 Сценарий ручной разметки

Инициатором данного сценария является эксперт. По мере работы эксперта в системе шаги данного сценария циклически повторяются. На рисунке 5 приведена одна итерация сценария ручной разметки. Графический интерфейс приложения и API системы на схеме опущены для упрощения диаграммы.

Кратко опишем процесс ручной разметки с точки зрения взаимодействия различных компонентов системы.

1. Первым делом эксперт отправляет запрос на получение задания. Под заданием в данном случае может подразумеваться как задание разметки данных, так и задание валидации автоматической разметки, так как для двух указанных случаев шаги сценария будут одинаковыми.

2. *MarkupService* обрабатывает запрос, и вызывает *TaskAssignmentService* для получения назначения задания (сущность *Appointment*).

3. *TaskAssignmentService* проверяет наличие в базе назначения для заданного пользователя.

4. Если назначения в базе не было найдено, тогда:

    4.1. *TaskAssignmentService* определяет то, какой элемент данных эксперт должен разметить следующим. Для определения следующего элемента данных для разметки *TaskAssignmentService*, в общем

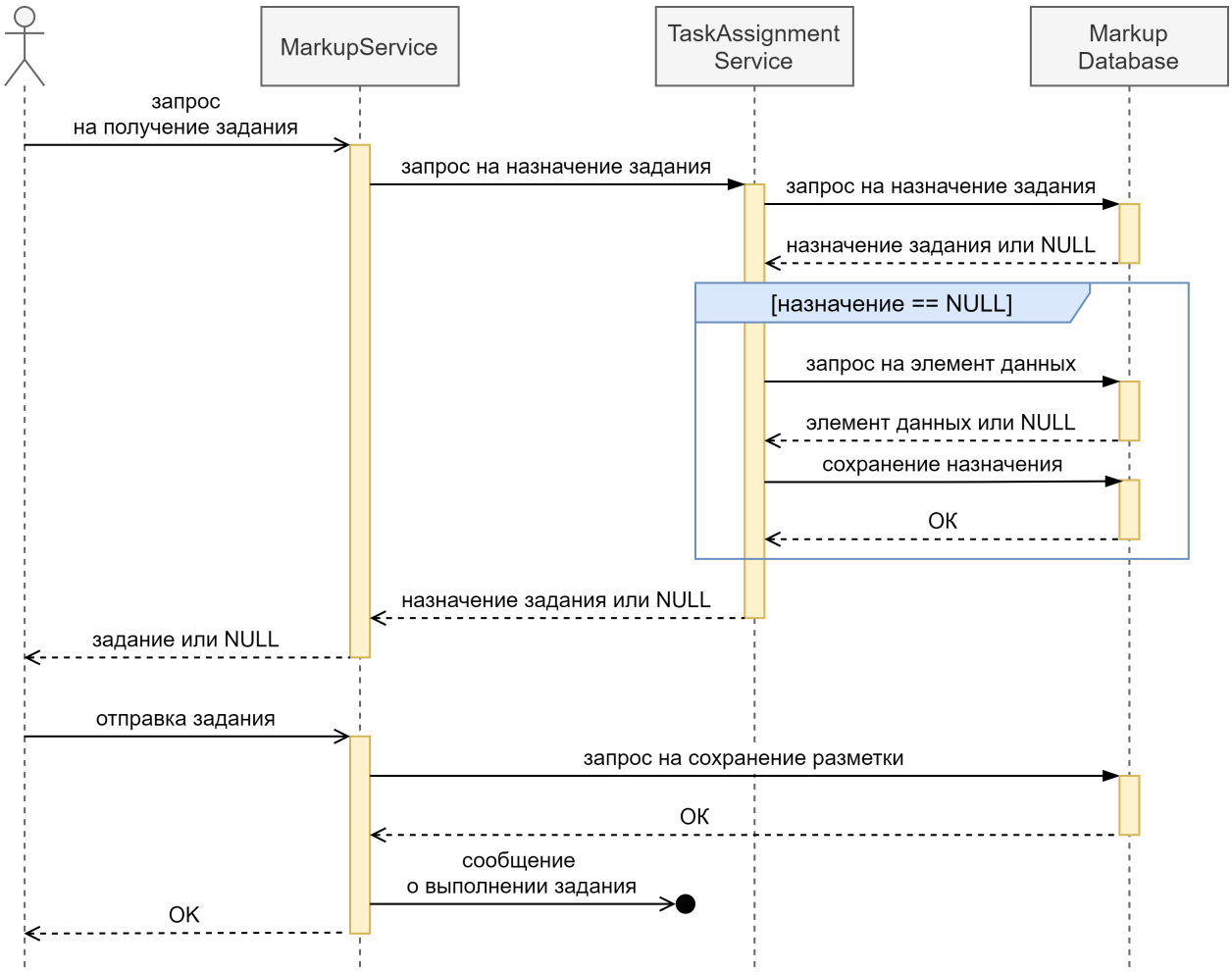


Рис. 5: Диаграмма последовательности сценария ручной разметки.

случае, может использовать несколько запросов к *MarkupDatabase*, а также к локальной базе данных. Для упрощения схемы на диаграмме показан один запрос к *MarkupDatabase* для получения следующего элемента данных.

4.2. В случае, если элемент данных, который должен разметить эксперт, нашелся, то формируется назначение и сохраняется в *MarkupDatabase*.

5. Назначение возвращается в *MarkupService*.

6. *MarkupService* возвращает пользователю задание, которое необходимо выполнить.

7. Пользователь выполняет задание и отправляет результат в систему.

8. *MarkupService* сохраняет ответ пользователя в *MarkupDatabase*.

9. *MarkupService* генерирует сообщение о том, что было выполнено некоторое задание. Данное сообщение обрабатывается остальными компонентами в системы в рамках других сценариев.

10. *MarkupService* возвращает ответ пользователю об успешном сохранении результата.

3.3.2 Сценарий обучения модели

На рисунке 6 приведено схематичное изображение данного сценария.

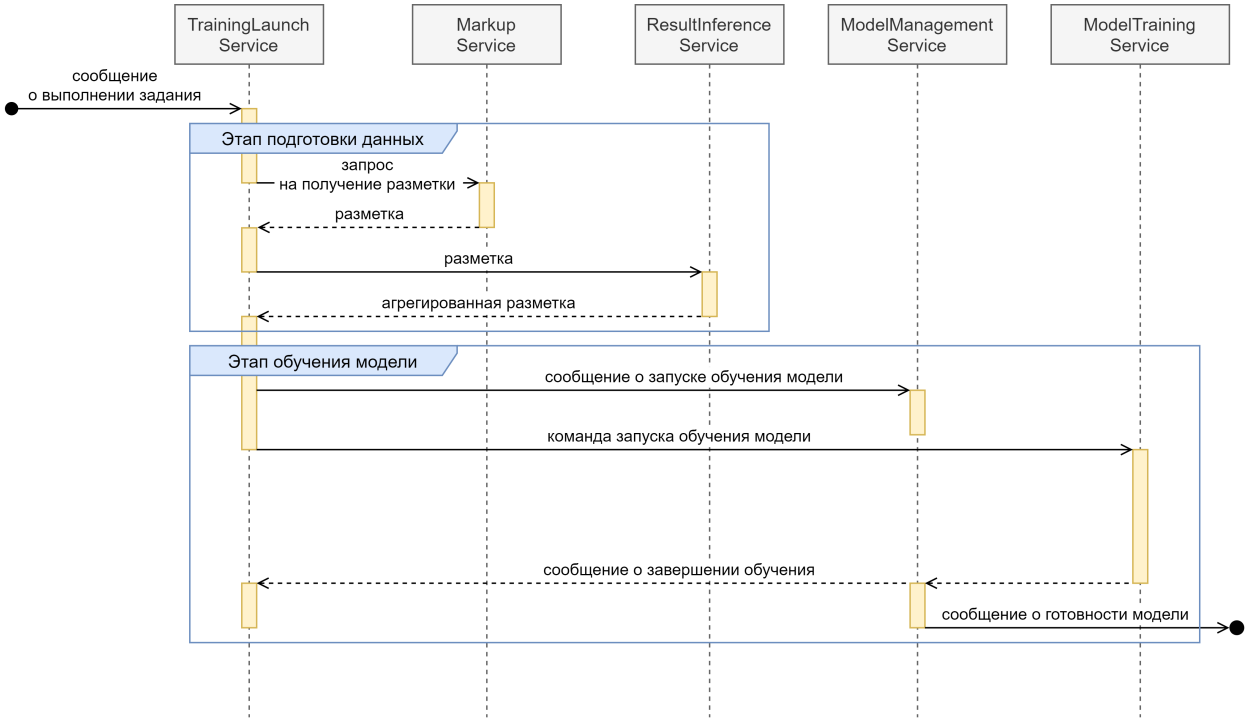


Рис. 6: Диаграмма последовательности сценария обучения модели.

Сообщение о выполнении задания экспертом инициирует выполнение сценария обучения модели. Данное решение обосновывается тем, что обучение модели должно производиться по мере накопления размеченных данных в системе, при этом сообщение о выполнении задания экспертом прямо указывает на изменение данных. Альтернативным решением был бы запуск сценария обучения модели по прошествии определенного времени, либо же

в заданные моменты времени, однако в таком случае нет гарантии того, что данные в системе были изменены.

Также предполагается, что *TrainingLaunchService* в целях оптимизации может пропускать некоторые сообщения о выполнении задания, чтобы не запускать процесс обучения модели при незначительном увеличении объема размеченных данных.

Сценарий обучения модели может быть условно разделен на два этапа:

1. этап подготовки данных;
2. этап обучения модели.

Этап подготовки данных начинается в тот момент, когда *TrainingLaunchService* получает сообщение о выполнении задания и принимает решение о том, что следует начать процесс обучения модели. Этап состоит из следующих шагов:

1. *TrainingLaunchService* отправляет запрос на получение размеченных данных в *MarkupService*.
2. *MarkupService* выгружает из *MarkupDatabase* разметку пользователей, автоматическую разметку и результаты валидации автоматической разметки, и отправляет *TrainingLaunchService*.
3. *TrainingLaunchService* отправляет разметку в *ResultInferenceService*, который осуществляет агрегацию разметки и возвращает результат. На данном шаге завершается этап подготовки данных.

Этап обучения модели включает в себя следующие шаги:

1. *TrainingLaunchService* отправляет *ModelManagementService* сообщение о том, что обучение модели запустилось. *ModelManagementService* создает запись в *ModelDatabase* о новой модели и помечает ее как «в процессе».
2. *TrainingLaunchService* отправляет команду *ModelTraining* запустить обучение модели. Вместе с командой сервис также отправляет предобработанные данные и информацию о типе задачи разметки.

3. *ModelTrainingService* осуществляет обучение модели и, в случае успешного завершения, отправляет сообщение об успешном завершении операции. Данный сервис также сохраняет веса в *WeightsVolume*, а путь к ним прикрепляет в сообщение об успешном завершении обучения модели.

4. *ModelManagementService* обрабатывает сообщение о завершении обучения и обновляет запись о модели в базе, изменяя статус модели на «готово» и указывая путь к весам модели.

5. После обновления базы данных, *ModelManagementService* генерирует сообщение о том, что модель обучена и может быть использована для автоматической разметки.

3.3.3 Сценарий автоматической разметки

Графическое представление сценария автоматической разметки приведено на рисунке 7.

Данный сценарий состоит двух этапов:

1. Проверка условия запуска автоматической разметки.

2. Автоматическая разметка.

Сценарий автоматической разметки изображений может запускаться в двух следующих случаях:

- пришло сообщение о том, что было выполнено некоторое задание валидации модели;

- пришло сообщение о том, что была обучена некоторая модель.

Сценарий может быть условно разделен на два этапа: проверку условия запуска автоматической разметки и непосредственно автоматическую разметку.

Проверка условия запуска автоматической разметки является опциональным этапом. В случае, если пришло сообщение о готовности модели

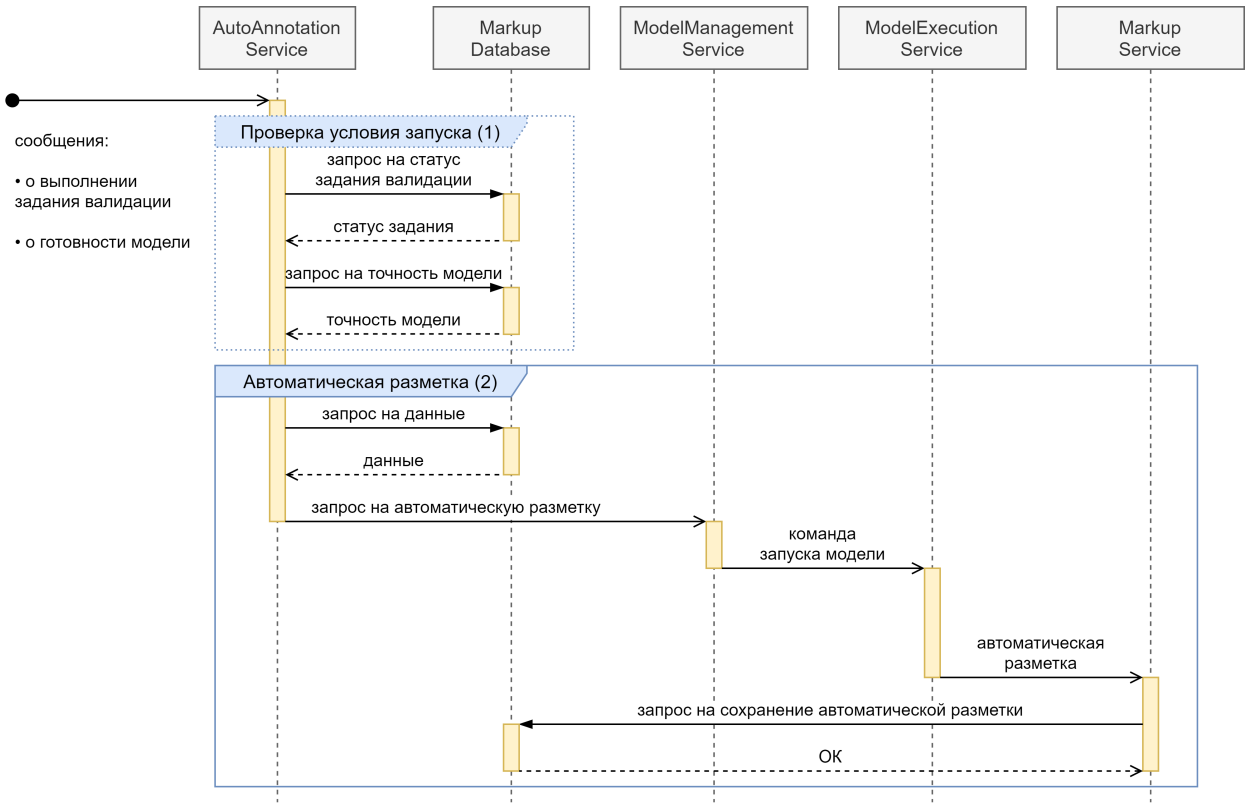


Рис. 7: Диаграмма последовательности сценария автоматической разметки.

машинного обучения, данный этап пропускается. Действительно, в случае, когда в системе была обучена новая модель, имеет смысл произвести автоматическую разметку с ее использованием для последующей оценки точности этой модели. Данный этап состоит из следующих шагов:

1. *AutoAnnotationService* осуществляет проверку того, были ли проверены результаты предыдущего блока автоматически размеченных элементов данных. Для этого сервис осуществляет запрос к *MarkupDatabase*.

2. В случае, если экспертами были выполнены не все задания валидации автоматической разметки, тогда *AutoAnnotationService* прекращает выполнение сценария. Если же были выполнены все задания, тогда *AutoAnnotationService* оценивает точность модели на основе оценок экспертов, осуществляя запрос к *MarkupDatabase*.

3. Если точность оказалась достаточно низкой, тогда *AutoAnnotationService* прекращает выполнение сценария. То, как оценивается точность модели

и то, как задается необходимая точность, зависит от реализации описываемой архитектуры. В случае, если точность оказывается достаточной, начинается этап автоматической разметки.

Этап автоматической разметки состоит из следующих шагов:

1. *AutoAnnotationService* получает некоторое количество (зависит от реализации) элементов данных для автоматической разметки, осуществляя запросы к *MarkupDatabase*.

2. Далее *AutoAnnotationService* отправляет запрос на автоматическую разметку с использованием последней актуальной обученной модели в *ModelManagementService*.

3. *ModelManagementService*, используя базу данных *ModelDatabase*, находит последнюю актуальную модель, а также необходимую для ее запуска метаинформацию.

4. *ModelManagementService* отправляет команду запуска *ModelExecutionService*, а также передает данные, которые необходимо разметить, и метаинформацию о модели.

5. *ModelExecutionService* осуществляет разметку переданных данных с помощью требуемой модели и передает результат в *MarkupService*, который далее сохраняет автоматически размеченные данные в базе.

3.4 Отказоустойчивость приложения

В данном разделе приводится анализ устойчивости разработанной архитектуры приложения ко сбоям в работе различных компонентов системы.

Прежде всего отметим, что для работы системы критична работоспособность *MarkupService*: в случае сбоя в работе данного сервиса, функционал приложения будет недоступен. Однако данный сервис не содержит сложных вычислений, из-за которых потенциально могут возникать сбои в его работе. В случае, если происходит сбой в любых других микросервисах приложения, а *MarkupService* остается рабочим, то, как минимум, система будет предоставлять пользователям функционал по ручной разметке.

В случае, если происходит сбой в сервисах *ResultInferenceService* или *TrainingLaunchService*, тогда для тех заданий разметки, для которых уже были обучены модели, сценарий автоматической разметки будет продолжать выполняться в обычном режиме, однако обучение новых моделей будет приостановлено.

Если предположить, что выйдет из строя *ModelExecutionService* или *AutoAnnotationService*, то сценарий автоматической разметки будет недоступен, однако в системе будут продолжать обучаться модели, и после устранения сбоя сценарий автоматической разметки будет использовать новые версии моделей.

В случае, если сбой происходит в *ModelManagementService*, то сценарий обучения моделей и сценарий автоматической разметки не будут выполняться, но инструмент может быть использован для ручной разметки.

Таким образом, можно сделать вывод о том, что разработанная архитектура системы является достаточно устойчивой к сбоям в работе различных компонентов. В большинстве случаев при возникновении сбоев пользователи системы смогут использовать инструмент для осуществления ручной разметки, и работа по разметке данных будет продолжаться. Наиболее уязвимым местом архитектуры является *MarkupService*, однако это компенсируется отсутствием в нем сложных вычислений, из-за которых могут происходить сбои в работе сервиса.

Глава 4. Разработка прототипа

В качестве подтверждения работоспособности предложенной архитектуры инструмента разметки данных был разработан прототип приложения. В данной главе рассматриваются детали реализации разработанного прототипа.

Репозиторий проекта размещен в GitHub и доступен по следующей ссылке: https://github.com/Godis715/Markup-Tool.

4.1 Ограничения прототипа

При разработке прототипа в качестве задач, для которых требуется подготовка размеченных данных, рассматривались задачи из области машинного зрения. Данное ограничение области рассматриваемых задач связано с большой распространенностью задач машинного зрения в различных областях человеческой деятельности. Выбор области решаемых задач для разработки прототипа был не принципиален, так как архитектура разработанного инструмента позволяет масштабировать приложение и на другие сферы применения машинного обучения, такие как обработка естественного языка и пр.

Для демонстрации возможности масштабирования системы на множество задач разметки в прототипе была реализована возможность разметки нескольких типов задач. Согласно [1], в качестве базовых задач разметки следует использовать достаточно простые задачи, которые могут быть быстро решены экспертами. При этом более сложные задачи разметки должны представляться в виде последовательного решения нескольких базовых задач. В соответствие с этим принципом, в прототипе были реализованы следующие типы разметки изображений:

- Классификация изображений. Для разметки изображения эксперту требуется указать один из перечисленных классов.
- Локализация объекта на изображении. Для разметки изображения эксперту требуется выделить требуемый объект на изображении с помощью прямоугольника (ограничивающей рамки).
- Локализация множества объектов на изображении. Для разметки изоб-

ражения эксперту требуется выделить все объекты требуемого типа на изображении с помощью прямоугольников.

Заметим, что задача детекции объектов на изображении, заключающаяся в выделении объектов на изображении прямоугольной рамкой и указании классов объектов, может быть решена с использованием реализованных в инструменте базовых типов разметки: первым делом эксперты должны выделить прямоугольными рамками требуемые объекты на изображении, а затем эксперты должны осуществить классификацию объектов внутри выделенных рамок.

Сценарий автоматической разметки в прототипе был реализован для задачи локализации множества объектов, так как данный тип задачи является наиболее трудоемким среди реализованных типов разметки.

4.2 Технологический стек

Целевой программной платформой для реализации инструмента разметки данных была выбрана Web-платформа. Основными достоинствами реализации Web-приложения в данной работе являются отсутствие потребности в установке приложения с точки зрения пользователей, а также кроссплатформенность.

В качестве изолированного окружения для работы микросервисов приложения были выбраны Docker-контейнеры. Docker позволяет определять сценарии инициализации и запуска микросервисов, обеспечивает изолированную работу компонентов приложения внутри контейнеров, а также предоставляет базовую функциональность по управлению контейнерами.

В качестве инструмента для сборки и запуска всего приложения была выбрана утилита Docker Compose. Данная утилита позволяет описывать основные сценарии сборки и запуска приложения, настраивать способы коммуникации между контейнерами.

Для реализации сервисов приложения, реализующих алгоритмы чтения и записи в различные базы данных, а также простые вычисления, была выбрана программная платформа NodeJS. NodeJS позволяет реализовывать веб-сервисы с большой пропускной способностью за счет использования асин-

хронных вычислений [22]. Для написания NodeJS-сервисов использовался язык TypeScript. Для реализации API серверной части NodeJS-приложения был использован веб-фреймворк Express[6]. Для проверки корректности данных в формате JSON была выбрана библиотека Ajv JSON schema validator[7].

Компоненты подсистемы автоматической разметки данных, осуществляющие обработку данных, обучение и запуск моделей, было решено реализовывать с использованием языка Python. Основным аргументом для использования Python был тот факт, что данный язык имеет большое количество реализованных библиотек для работы с данными и алгоритмами машинного обучения.

Для обеспечения коммуникации микросервисов посредством сообщений была использована очередь сообщений RabbitMQ. Основным аргументом в пользу использования RabbitMQ стало то, что данный сервис позволяет как организовывать коммуникацию между мкиросервисами, так и создавать очереди задач, которые необходимы при наличии ресурсоемких вычислений в системе [23]. В качестве клиентских библиотек для коммуникации между микросервисами с использованием RabbitMQ для NodeJS и Python использовались соответственно amqplib[8] и pika [9].

Было решено использовать реляционную модель для хранения постоянных данных. Выбор конкретной СУБД в данной работе был не принципиален, и в качестве одной из возможных СУБД была выбрана PostgreSQL[10]. В качестве ORM в NodeJS-сервисах использовался фреймворк TypeORM[11]. Использование ORM в данной работе позволяет абстрагировать алгоритмы работы микросервисов от деталей функционирования выбранной СУБД. Также использование ORM позволяет достаточно просто переключиться на другую СУБД, аналогичную PostgreSQL.

Для реализации клиентской части приложения был выбран фреймворк для создания графических веб-интерфейсов React[12].

[6]https://github.com/expressjs/express
[7]https://github.com/ajv-validator/ajv
[8]https://github.com/squaremo/amqp.node
[9]https://github.com/pika/pika
[10]https://www.postgresql.org/
[11]https://github.com/typeorm/typeorm
[12]https://reactjs.org/

4.3 Протоколы коммуникации

Для коммуникации клиентской и серверной части приложения был разработан REST API. Пример реализованных методов API приведен в приложении 1. Полная документация разработанного API доступна по ссылке: https://github.com/Godis715/Markup-Tool/wiki/API-documentation.

В качестве базового протокола коммуникации между разработанными микросервисами используется протокол передачи сообщений AMQP [24]. Микросервисы взаимодействуют посредством передачи друг другу сообщений в формате JSON. Описание типов сообщений приведено в приложении 2. Документация протоколов коммуникации микросервисов доступна по ссылке: https://github.com/Godis715/Markup-Tool/wiki/Messages-documentation.

4.4 Алгоритм назначения заданий

Зачастую при разработке систем разметки данных используется простейший способ назначения заданий экспертам: случайный выбор из множества доступных заданий. Однако существуют более продвинутые техники выбора элементов данных, которые следует разметить в первую очередь. Одной из таких техник является активное обучение, позволяющее выбирать неразмеченный элемент данных, разметка которого позволит значительно увеличить точность модели машинного обучения при его включении в обучающую выборку. Однако активное обучение – это лишь одна из возможных стратегий выбора следующего для разметки элемента.

Заметим также, что при использовании машинного обучения для автоматизации процесса разметки, требуется накопление большого объема данных. С другой стороны, для повышения качества данных требуется производить разметку одного элемента данных сразу несколькими экспертами [6]. В данном случае снова возникает неоднозначность при назначении задания эксперту: следует ли ему размечать еще не обработанное никем изображение, либо же он должен выполнить разметку уже обработанного другими экспертами изображения, чтобы повысить точность агрегированной разметки.

Можно заметить, что множество всех элементов данных может быть разбито на некоторые «группы интересов», например, элементы выбранные

приоритетными с помощью активного обучения, или элементы, для которых требуется дополнительная разметка из-за неоднозначности в ответах экспертов. При этом нельзя однозначно сказать, элементы из какой группы требуется размечать в первую очередь. При реализации прототипа данная проблема была решена предложением достаточно общего алгоритма назначения заданий, предполагающего наличие нескольких «групп интересов» в наборе элементов данных, которые требуется разметить.

Пусть E – множество всех элементов данных. Обозначим как $G_1, ..., G_n$, $n > 0$ группы элементов данных. При этом $E = \bigcup_i G_i$. Также будем предполагать, что группы могут пересекаться.

Пусть $queryElement(E_i, G_j)$ – функция, позволяющая для эксперта E_i и группы G_j получить элемент данных $a \in G_j$, который не размечался данным экспертом. Если же такого элемента не нашлось, то функция возвращает специальное значение $NULL$.

Предположим, что также задан вектор вероятностей $Probab$:

$$Probab = (p_1, ..., p_n), p_i \geq 0, i = \overline{1, n} \sum p_i = 1.$$

Пусть $randomIndex(P)$ – функция, которая принимает вектор вероятностей $P = (p_1, ..., p_n)$ и возвращает число i с вероятностью p_i.

Запишем следующий алгоритм $getNextElement(E_i)$ назначения задания, который для заданного эксперта E_i возвращает элемент данных для разметки.

Function `getNextElement(`E_i`)`**:**

$P \leftarrow Probab$
while $\exists P[j] > 0$ **do**
$k \leftarrow randomIndex(P)$
$a \leftarrow queryElement(E_i, G_k)$
if $a \neq NULL$ **then**
return a
if $P[k] \neq 1$ **then**
$P \leftarrow \frac{P}{1-P[k]}$
$P[k] \leftarrow 0$
return *NULL*

Алгоритм 1: Алгоритм назначения заданий

Интерпретация алгоритма заключается в следующем: если для данного эксперта доступны задания из всех групп, тогда эксперт получит задание из группы G_i с вероятностью p_i.

Для использования приведенного алгоритма на практике требуется выделить группы элементов данных и для каждой из этих групп реализовать функцию $queryElement$. Данная функция может представлять собой некоторый SQL-запрос к базе, в которой хранятся элементы данных. Данный алгоритм также требует задания вектора вероятностей $Probab$ для выделенных групп элементов. Данный вектор может быть статически заданным, либо же определяться заказчиком, который формирует задание разметки. При формировании вектора вероятностей следует опираться на предполагаемую приоритетность элементов данных из заданной группы для разметки. Например, если более приоритетным является как можно более быстрое накопление размеченных данных, то для элементов данных, которые еще не были размечены, стоит указать большую вероятность выбора. Если же целью является формирование как можно более точной разметки, то большую вероятность выбора стоит задавать элементам данных, на которых была получена противоречивая экспертная разметка.

Таким образом, достоинствами предложенного алгоритма являются:

- Гибкость. Задание вектора вероятности позволяет контролировать среднюю частоту выбора заданий из определенных групп.

- Масштабируемость. Для добавления новых групп элементов требуется только реализовать функцию $queryElement$ для получения элемента из группы для заданного эксперта.

В реализованном прототипе предполагается, что для каждого задания разметки установлена константа – минимальное количество экспертов, которые должны разметить один элемент данных. В случае, если элемент данных был размечен заданным или бóльшим количеством экспертов, разметка данного элемента больше не производится.

В прототипе инструмента разметки были выделены следующие группы элементов данных:

1. Элементы, которые не были размечены ни одним экспертом.

2. Элементы, которые были размечены хотя бы одним экспертом и при этом количество таких экспертов меньше заданной константы.

Функция $queryElement$ была реализована как группа SQL-запросов к *MarkupDatabase*. Каждому SQL-запросу соответствует одна выделенная группа элементов данных.

4.5 Агрегация разметки

В разработанном прототипе инструмента разметки данных был реализован алгоритм агрегации результатов локализации множества объектов на изображении. На рисунке 8а приведены примеры экспертной разметки, которую требуется преобразовать в итоговую разметку.

Как можно видеть, эксперты размечают требуемые объекты, однако рамки вокруг объектов могут не совпадать, некоторые эксперты могут ошибочно отмечать лишние объекты. Возможны ситуации, когда эксперты будут ошибочно или намеренно создавать неверную разметку. Алгоритм, идея которого описана в статье [15], позволяет решить следующие задачи при агрегации результатов локализации множества объектов:

1. Определить количество объектов на изображении.

2. Отсеять аномалии в разметке пользователей.

3. Получить ограничивающие рамки объектов на изображении.

Базовый алгоритм, предложенный авторами статьи, состоит из двух этапов:

1. Выделение кластеров на множестве ограничивающих рамок, отмеченных экспертами.

2. Объединение ограничивающих рамок внутри одного кластера в один прямоугольник.

Для выделения кластеров ограничивающих рамок авторы статьи предлагают использовать алгоритм DBSCAN, впервые предложенный в статье [25]. Как отмечают авторы статьи, данный алгоритм позволяет произвести разбиение всего множества ограничивающих рамок без заранее известного количества объектов на изображении, а также отсеять аномалии.

Алгоритм DBSCAN требует задания функции расстояния (или функции близости) на множестве элементов, которые требуется кластеризовать. DBSCAN имеет два параметра – ϵ, $minPts$. ϵ – диаметр окрестности, в которой осуществляется поиск соседних объектов, $minPts$ – минимальное количество соседних точек, необходимых для образования кластера.

В качестве функции близости на множестве ограничивающих рамок авторы статьи предлагают использовать площадь пересечения двух рамок. Способ подбора параметров DBSCAN в статье не описывается, поэтому при реализации прототипа был предложен следующий вариант:

- Значение $minPts$ может быть рассмотрено как предполагаемое количество экспертов, которые должны разметить объект на изображении, чтоб данный объект вошел в итоговую разметку. Тогда $minPts$ можно положить как $minPts = \lfloor q \cdot expertsNum \rfloor$, где $expertsNum$ – общее количество экспертов, разметивших изображение, $q \in (0, 1]$ – некоторое пороговое значение, например, 0.75.

- Значение ϵ стоит подобрать эмпирическим путем и зафиксировать.

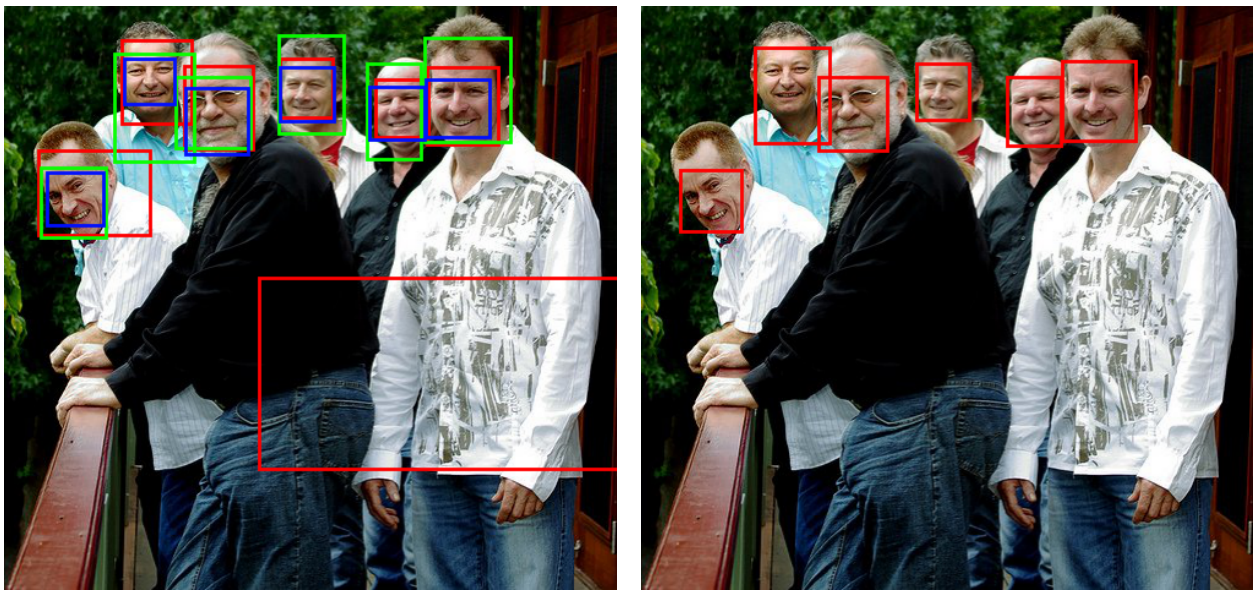
Отметим, что предлагаемый авторами статьи вариант выбора функции расстояния на множестве ограничивающих рамок имеет проблему того, что расстояние между рамками будет изменяться при масштабировании координат ограничивающих рамок. По этой причине выбор ϵ, учитывающий масштаб изображения, становится достаточно трудной задачей.

В качестве решения проблемы в прототипе в качестве функции близости использована функция Intersection over Union (IoU). Достоинством данной функции является инвариантность относительно масштабирования [26].

После того как произведено разбиение ограничивающих рамок на кластеры, осуществляется процесс объединения ограничивающих рамок внутри кластеров. Авторы статьи в качестве базового подхода предлагают использовать процедуру усреднения по координатам. Предположим, в кластер попали рамки, заданные через координаты двух противоположных углов: $r_i = (x^i_{min}, y^i_{min}, x^i_{max}, y^i_{max})$, $x^i_{min} \leq x^i_{max}$, $y^i_{min} \leq y^i_{max}$, $i = \overline{1, n}$, $n > 0$. Тогда координаты итоговой ограничивающей рамки R определяются следующим образом:

$$R = \frac{1}{n} \sum_{i=1}^{n} r_i$$

Результаты работы реализованного алгоритма приведены на рисунке 8.



(a) Разметка трех пользователей **(b)** Результат агрегации с параметрами $q = 0.75, \epsilon = 0.5$

Рис. 8: Пример работы алгоритма вывода разметки для задания локализации множества объектов.

4.6 Автоматическая разметка

Для реализации автоматической локализации множества объектов были решено использовать популярную архитектуру нейронных сетей для детекции объектов на изображении YOLO. Нейронные сети семейства YOLO демонстрируют высокую точность детекции объектов в классе моделей, работающих в режиме реального времени [10]. Отметим, что в данной работе выбор конкретной модели для реализации системы автоматической разметки не принципиален, потому что программная архитектура не накладывает ограничений на используемую модель для автоматической разметки.

В качестве используемой реализации модели семейства YOLO была использована модель YOLOv5s архитектуры YOLOv5 [27]. Данная архитектура не является разработкой авторов предыдущих версий YOLO и на данный момент находится в активной разработке, однако так как данная модель является наиболее современной, было решено для реализации прототипа использовать ее. Отметим, что принцип использования моделей семейства YOLO один и тот же, и выбранная модель может быть заменена другой.

Модели архитектуры YOLOv5 решают задачу детекции объектов на изображении, поэтому для применения их в приложении, требовалось свести

задачу детекции к задаче локализации множества объектов. Наиболее простым способом сведения одной задачи к другой является представление задачи локализации множества объектов как задачи детекции с количеством классов, равным 1. Для обучения модели на заданном наборе данных использовался Python-скрипт `train.py`[13], реализованный авторами модели, с параметрами по умолчанию. В качестве стартовых весов модели в реализованном прототипе используются веса обученной на наборе данных COCO[14] модели YOLOv5s. Полученные в результате обучения модели веса сохраняются в *WeightsVolume* и в дальнейшем используются для автоматической разметки изображений сервисом *AutoAnnotationService*.

4.7 Демонстрация разработанного приложения

Изображения, демонстрирующие основную функциональность прототипа приведены в приложении 3.

[13]https://github.com/ultralytics/yolov5/blob/develop/train.py

[14]https://cocodataset.org/#home

Глава 5. Анализ разработанного прототипа

В данном разделе приводится теоретическая оценка эффективности разработанного приложения, а также методология и результаты тестирования приложения.

5.1 Теоретическая оценка ускорения процесса ручной разметки

В данном подразделе дается теоретическая оценка ускорения процесса ручной разметки при использовании разработанного прототипа.

Рассмотрим следующий упрощенный (по сравнению с реализованным) алгоритм полуавтоматической разметки, который состоит из пяти последовательных этапов:

1. Разметка экспертами тренировочного набора данных.
2. Обучение модели на тренировочном наборе данных.
3. Прогнозирование разметки для каждого неразмеченного элемента данных моделью.
4. Ручная валидация результатов работы модели.
5. Разметка экспертами тех изображений, результаты автоматической разметки которых были отклонены на этапе валидации.

В данном алгоритме сделаны следующие упрощения, по сравнению с алгоритмом полуавтоматической разметки, реализованным в прототипе:

- Модель обучается ровно один раз. В разработанном прототипе модель может обучаться несколько раз по мере накопления данных.
- Все этапы идут последовательно. В разработанном прототипе обучение модели может идти параллельно процессу ручной разметки.
- Модель размечает все неразмеченные изображения из набора данных, а пользователи валидируют всю разметку, которую произвела модель. В реализованном прототипе в случае, если модель демонстрирует низкую точность, автоматическая разметка не производится.

Введем следующие переменные:

- $N \geq 0$ – количество изображений, которые необходимо разметить;
- $M > 0$ – кол-во экспертов;
- $t_{exp} > 0$ -– время разметки изображения одним экспертом;
- $L \in (0, N]$ -– требуемое для обучения кол-во изображений;
- $T(x) > 0$ -– время обучения в зависимости от количества изображений x в обучающей выборке;
- $p \in [0, 1]$ — доля изображений, которые эксперты оценят как верные;
- $t_{val} > 0$ -– время валидации автоматической разметки одним экспертом;
- $t_{model} > 0$ — время разметки изображения моделью;
- $m \in (0, M]$ — требуемое количество ручных разметок одного изображения;
- $v \in (0, M]$ -– требуемое количество валидаций автоматической разметки.

На основании введенных переменных введем несколько вспомогательных формул:

- $T_{manual} = \frac{m}{M} N t_{exp}$ — время разметки набора данных вручную (M экспертов работают параллельно и независимо);
- $T_1 = \frac{m}{M} L t_{exp}$ -– время разметки тренировочного набора данных для модели (первый этап алгоритма);
- $T_2 = T(L)$ — время обучения модели (второй этап алгоритма);
- $T_3 = (N - L) t_{model}$ — длительность третьего этапа;
- $T_4 = (N - L) \frac{v}{M} t_{val}$ — время, которое потребуется на валидацию предсказаний модели (четвертый этап);

- $(N - L)(1 - p)$ — количество изображений, которые модель разметит неверно (по мнению экспертов);
- $T_5(p) = \frac{m}{M}(N-L)(1-p)t_{exp}$ — время на разметку изображений, которые, по мнению экспертов, были размечены моделью неверно (пятый этап). Длительноть данного этапа запишем как функцию от экспертной оценки точности модели p.

Запишем итоговую длительности процесса полуавтоматической разметки T_{auto} как функцию от p:

$$T_{auto}(p) = T_1 + T_2 + T_3 + T_4 + T_5(p).$$

Ускорение процесса разметки при использовании алгоритма полуавтоматической разметки выражается следующим образом:

$$\triangle t(p) = T_{manual} - T_{auto}(p). \tag{1}$$

Заметим, что величина $T_5(p)$ неотрицательна в силу ограничений, накладываемых на переменные. $T_5(p)$ принимает минимальное значение при $p = 1$ и максимальное при $p = 0$. Запишем максимальное и минимальное значение ускорения процесса ручной разметки:

$$\triangle t_{min} = T_{manual} - T_{auto}(0) = -(T_2 + T_3 + T_4),$$
$$\triangle t_{max} = T_{manual} - T_{auto}(1) = T_{manual} - (T_1 + T_2 + T_3 + T_4).$$

Заметим, что использование полуавтоматической разметки может быть оправдано только в том случае, когда $\triangle t_{max} \geq 0$. Преобразуя данное неравенство, получим условие применимости полуавтоматической разметки:

$$mt_{exp} \geq Mt_{model} + vt_{val} + \frac{MT(L)}{N - L}.$$

Далее рассмотрим следующую задачу: при заданных значениях всех переменных, кроме p, определим значение p, при котором ускорение процесса разметки будет равняться нулю. Иными словами, определим минимальную

необходимую точность модели, чтобы процесс полуавтоматической разметки был не дольше, чем процесс ручной разметки.

Используя формулу 1, найдем значение p_0, при котором $\triangle t(p_0) = 0$:

$$p_0 = \frac{1}{mt_{exp}} \left(Mt_{model} + vt_{val} + \frac{MT(L)}{N - L} \right).$$

Нетрудно заметить, что при выполнении условия $\triangle t_{max} \geq 0$ значение p_0 не превосходит 1.

Далее зададим исходным переменным значения, отвечающие некоторым реальным ситуациям, которые могут возникнуть в задачах разметки данных.

Предположим, что имеется лишь 1 эксперт ($M = 1$), размечающий набор данных, состоящий $N = 10^5$ изображений. Количество размеченных изображений, необходимых для обучения модели положим равным $L = 1000$. Пусть эксперт размечает одно изображение $t_{exp} = 30$ секунд, а валидирует разметку одного изображения $t_{val} = 5$ секунд. Время обучения модели на x изображениях положим равным $T(x) = 10x + 3600$ секунд. Время разметки одного изображения моделью положим равным $t_{model} = 0.05$ секунд, что является реалистичным значением для некоторых современных моделей машинного обучения и современных вычислительных устройств.

При сделанных предположениях можем найти $\triangle t_{min}$, $\triangle t_{max}$, p_0:

$$\triangle t_{min} \approx -143 \text{ ч.}, \triangle t_{max} \approx 682 \text{ ч.}, \triangle p_0 \approx 0.17.$$

Далее предположим, что экспертов теперь $M = 5$, требуется, чтобы каждое изображение было размечено ровно $m = 1$ экспертами, а каждый прогноз модели должен быть проверен ровно $v = 3$ экспертами. Все остальные параметры оставим неизменными.

$$\triangle t_{min} \approx -88 \text{ ч.}, \triangle t_{max} \approx 77 \text{ ч.}, \triangle p_0 \approx 0.53.$$

В случае, если требуется разметка изображений сразу $m = 3$ экспертами,

то получаем следующее:

$$\triangle t_{min} \approx -88 \text{ ч.}, \triangle t_{max} \approx 407 \text{ ч.}, \triangle p_0 \approx 0.18.$$

Как можно видеть, для указанных значений параметров в рамках рассматриваемого алгоритма полуавтоматической разметки изображений, требуются достаточно реалистичные значения точности моделей, необходимой для ускорения процесса разметки. В рассмотренных примерах можно видеть, что использование упрощенного алгоритма полуавтоматической разметки в лучшем случае может давать значительно ускорение. При этом можно видеть, что в худшем случае алгоритм полуавтоматической разметки замедляет процесс на время, равное суммарному времени, затраченному на обучение модели, разметку изображений моделью и валидацию прогнозов модели экспертами. На практике указанные временные затраты в худшем случае могут быть уменьшены за счет:

- параллельного обучения модели и разметки изображений пользователями;
- прекращения процесса валидации и перехода на ручную разметку в случае, если модель демонстрирует низкую точность;
- периодического переобучения модели по мере накопления размеченных данных для повышения точности прогнозирования.

5.2 Тестирование приложения

Для проверки работоспособности приложения были использованы два подхода тестирования: ручное тестирование и модульное тестирование. Во время ручного тестирования в запущенном приложении проверялось выполнение функциональных требований, описанных в разделе 2.3.

Для контроля корректности работы отдельных расчетных функций в приложении использовались модульные тесты, осуществляющие сравнение предполагаемых выходных данных функций и фактических на заданном наборе входных значений.

Вывод

Разработанный прототип инструмента для автоматизированной разметки данных продемонстрировал состоятельность предложенной программной архитектуры. В разделе 5.1 был приведен теоретический расчет возможного ускорения процесса ручной разметки с использованием разработанного прототипа. Для проверки функциональности приложения было произведено тестирование, методика которого описана в разделе 5.2.

Далее рассмотрим дальнейшие пути развития приложения. Прежде всего отметим, что расчеты, приведенные в разделе 5.1, могут лечь в основу системы, позволяющей динамически выбирать минимальную необходимую точность работы моделей для того, чтобы гарантировать ускорение процесса разметки. Динамически контролируя выбор порога точности модели, можно минимизировать риск того, что процесс полуавтоматической разметки будет длиться дольше, чем процесс ручной разметки, что было бы ценным качеством разрабатываемого инструмента.

Для оптимизации процесса автоматической разметки в приложении может быть реализована подсистема активного обучения. В качестве теоретической основы активного обучения могут служить работы, рассмотренные в обзоре литературы. Как можно видеть на примере сервиса Amazon SageMaker Ground Truth, активное обучение используется в коммерческом ПО для оптимизации процесса ручной разметки данных.

Спроектированная архитектура системы позволяет масштабировать инструмент для решения различных задач разметки данных. Благодаря этому, система в будущем может развиваться по пути добавления инструментов для разметки новых типов данных. Другим достоинством разработанной архитектуры приложения является устойчивость ко сбоям в работе различных компонентов системы.

Стоит отметить, что разработанная архитектура приложения учитывает возможность масштабирования приложения на множество узлов. Реализованный прототип запускает приложение на одном физическом сервере, однако архитектура автоматической разметки данных потенциально может быть масштабирована на множество узлов за счет использования инструментов

оркестрации контейнеров таких, как, например, Kubernetes. Достоинством создания распределенной системы может быть параллелизация таких ресурсоемких задач, как, например, обучение моделей, а также отделение их от сервиса разметки, который должен принимать и обрабатывать запросы пользователей в режиме высокой пропускной способности.

Заключение

В рамках данной работы была разработана архитектура инструмента для автоматизации процесса разметки наборов данных, которая обеспечивает масштабируемость и отказоустойчивость системы. На основе предложенной программной архитектуры был разработан прототип инструмента автоматизированной разметки данных. Осуществлен теоретический расчет ускорения, которое может обеспечить схема полуавтоматической разметки данных в реализованном прототипе. В выводе были отмечены основные направления развития разработанного приложения.

Список литературы

[1] Alexey Drutsa и др. «Practice of efficient data collection via crowdsourcing at large-scale». в: *arXiv* (2019), с. 1—9. ISSN: 23318422. arXiv: 1912.04444.

[2] Carsten Rother, Vladimir Kolmogorov и Andrew Blake. «"GrabCut Interactive foreground extraction using iterated graph cuts». в: *ACM Transactions on Graphics* 23.3 (2004), с. 309—314. ISSN: 07300301. DOI: 10.1145/1015706.1015720.

[3] Xuebin Qin и др. «ByLabel: A boundary based semi-automatic image annotation tool». в: *Proceedings - 2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018* 2018-Janua (2018), с. 1804—1813. DOI: 10.1109/WACV.2018.00200.

[4] Lluís Castrejón и др. «Annotating object instances with a polygon-RNN». в: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-Janua (2017), с. 4485—4493. DOI: 10.1109/CVPR.2017.477. arXiv: 1704.05548.

[5] Bryan C Russell и др. «LabelMe: A Database and Web-Based Tool for Image Annotation». в: *International Journal of Computer Vision* 77.1 (2008), с. 157—173. ISSN: 1573-1405. DOI: 10.1007/s11263-007-0090-8. URL: https://doi.org/10.1007/s11263-007-0090-8.

[6] R.A. Gilyazev и D.Y. Turdakov. «Active learning and crowdsourcing: a survey of annotation optimization methods». в: *Proceedings of the Institute for System Programming of the RAS* 30.2 (2018), с. 215—250. ISSN: 20798156. DOI: 10.15514/ispras-2018-30(2)-11.

[7] Nguyen Quoc Viet Hung и др. «An evaluation of aggregation techniques in crowdsourcing». в: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8181 LNCS.PART 2 (2013), с. 1—15. ISSN: 03029743. DOI: 10.1007/978-3-642-41154-0_1.

[8] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 2. University of Wisconsin–Madison, 2010, с. 201—221. DOI: 10.1 .1.167.4245.

[9] Elmar Haussmann и др. «Scalable Active Learning for Object Detection». в: *IEEE Intelligent Vehicles Symposium, Proceedings* (2020), с. 1430—1435. DOI: 10.1109/IV47402.2020.9304793. arXiv: 2004.04699.

[10] Joseph Redmon и др. «You only look once: Unified, real-time object detection». в: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem (2016), с. 779—788. ISSN: 10636919. DOI: 10.1109/CVPR.2016.91. arXiv: 1506.02640.

[11] Joseph Redmon и Ali Farhadi. «YOLO9000: Better, faster, stronger». в: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-Janua (2017), с. 6517—6525. DOI: 10.1109 /CVPR.2017.690. arXiv: 1612.08242.

[12] Joseph Redmon и Ali Farhadi. «YOLOv3: An Incremental Improvement». в: (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[13] Alexey Bochkovskiy, Chien-Yao Wang и Hong-Yuan Mark Liao. «YOLOv4: Optimal Speed and Accuracy of Object Detection». в: (2020). arXiv: 2004.1 0934. URL: http://arxiv.org/abs/2004.10934.

[14] Wei Liu и др. «SSD: Single shot multibox detector». в: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9905 LNCS (2016), с. 21—37. ISSN: 16113349. DOI: 10.1007/978-3-319-46448-0_2. arXiv: 1512.02325.

[15] Jocelyn C. Adams и др. «Grouper: Optimizing Crowdsourced Face Annotations». в: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* June (2016), с. 163—170. ISSN: 21607516. DOI: 10.1109/CVPRW.2016.27.

[16] *Amazon Mechanical Turk — Википедия*. URL: https://ru.wikipedia.org/wik i/Amazon_Mechanical_Turk (дата обр. 28.05.2021).

[17] *Toloka - Wikipedia*. URL: https://en.wikipedia.org/wiki/Toloka (дата обр. 29.05.2021).

[18] Boris Sekachev, Andrey Zhavoronkov и Nikita Manovich. *Computer Vision Annotation Tool: A Universal Approach to Data Annotation*. март 2019. URL: https://software.intel.com/content/www/us/en/develop/articles/computer-vision-annotation-tool-a-universal-approach-to-data-annotation.html (дата обр. 25.04.2021).

[19] *Automate Data Labeling - Amazon SageMaker*. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/sms-automated-labeling.html (дата обр. 29.05.2021).

[20] Karl E Wiegers и Joy Beatty. *Software Requirements 3*. USA: Microsoft Press, 2013. ISBN: 0735679665.

[21] Donald Bell. «UML basics: The sequence diagram». в: *Retrieved July* 17 (2004), с. 2015.

[22] *About | Node.js*. URL: https://nodejs.org/en/about/ (дата обр. 02.06.2021).

[23] *Messaging that just works — RabbitMQ*. URL: https://www.rabbitmq.com/ (дата обр. 30.05.2021).

[24] *AMQP 0-9-1 Model Explained — RabbitMQ*. URL: https://www.rabbitmq.com/tutorials/amqp-concepts.html (дата обр. 30.05.2021).

[25] M. Daszykowski и B. Walczak. «Density-Based Clustering Methods». в: *Comprehensive Chemometrics* 2 (2009), с. 635—654. DOI: 10.1016/B978-044452701-1.00067-3.

[26] Hamid Rezatofighi и др. «Generalized intersection over union: A metric and a loss for bounding box regression». в: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2019-June (2019), с. 658—666. ISSN: 10636919. DOI: 10.1109/CVPR.2019.00075. arXiv: 1902.09630.

[27] Glenn Jocher и др. *ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations*. 2021. DOI: 10.5281/zenodo.4679653. URL: https://doi.org/10.5281/zenodo.4679653.

Приложение 1. Примеры методов API

В данном приложении в качестве примеров методов реализованного API приводятся два метода для получения элемента для разметки и для отправки разметки. Полное описание API доступно по ссылке: https://github.com/Godis715/Markup-Tool/wiki/API-documentation.

GET `/api/task/:taskId/markup/item`

Метод для указанного задания разметки возвращает следующий элемент, который должен разметить эксперт. Содержимое ответа зависит от типа задания и, в случае с разметкой изображений, содержит ссылку на изображение.

URL-параметры:

- `taskId: string` – id задания.

Возвращает в случае успеха:

```
{
    "imageSrc": string
}
```

Коды ответов:

- 200 – в случае успеха;
- 401 – ошибка аутентификации;
- 403 – пользователь не является экспертом для данного задания;
- 404 – указано неверное задание или отсутствует элемент для разметки.

POST `/api/task/:taskId/markup/item`

Метод предназначен для отправки результатов разметки элемента, назначенного пользователю. Результат разметки представляет собой некоторый объект, структура которого зависит от решаемого задания.

URL-параметры:

- `taskId: string` – id задания.

Тело запроса:

- Если задание на классификацию объектов, то тело запроса включает в себя указание класса объекта в строковом виде:

```
{
    "result": string
}
```

- Если задание разметки представляет собой локализацию одного объекта на изображении, то тело запроса включает ограничивающих рамку объекта, либо указание того, что объект отсутствует:

```
{
    "result": {
        "status": "SUCCESS",
        "rectangle": {
            x1: number,
            y1: number,
            x2: number,
            y2: number
        }
    } | {
        "status": "CANNOT_DETECT_OBJECT"
    }
}
```

- Если задание на локализацию множества объектов, то тело запроса включает в себя перечисление ограничивающих рамок объектов на изображении, либо указание того, что объект отсутствует:

```
{
    "result": {
        "status": "SUCCESS",
```

```
            "rectangle": {
                x1: number,
                y1: number,
                x2: number,
                y2: number
            }[]
        } | {
            "status": "CANNOT_DETECT_OBJECT"
        }
    }
```

Коды ответов:

- 200 – ок;
- 400 – структура отправленной разметки не соответствует требуемой;
- 401 – ошибка аутентификации;
- 404 – задание не найдено или для пользователя нет назначения.

Приложение 2. Примеры сообщений

В данном приложении в качестве примера реализованной системы коммуникации между микросервисами приводятся два типа сообщений, которые используются для запроса автоматической разметки и уведомления о том, что автоматическая разметка была произведена. Полная документация сообщений, генерируемых микросервисами, доступна по ссылке: https://github.com/Godis715/Markup-Tool/wiki/RabbitMQ-messages-documentation.

Сообщение для запуска автоматической разметки в разработанном прототипе имеет следующий формат:

```
{
    "markupId": string,
    "markupType": string,
    "datasetItems": {
        "datasetItemId": string,
        "imageUrl": string
    }[]
}
```

Данное сообщение генерируется сервисом *AutoAnnotationService* и отправляется в очередь `model.predict` сервиса *ModelManagementService*, который запускает процесс автоматической разметки.

Сервис запуска моделей *ModelExecutionService* генерирует сообщение об успешном завершении автоматической разметки в следующем формате (приведен пример для задания локализации множества объектов на изображении):

```
{
    "markupId": string,
    "modelId": string,
    "markupType": string,
    "items": {
        "datasetItemId": string,
        "result": {
```

```
            "rectangles": {
                "x1": number,
                "y1": number,
                "x2": number,
                "y2": number
            }[]
        }
    }
}
```

Данное сообщение отправляется в очередь `model.predict.result` сервиса *MarkupService*.

Приложение 3. Демонстрация приложения

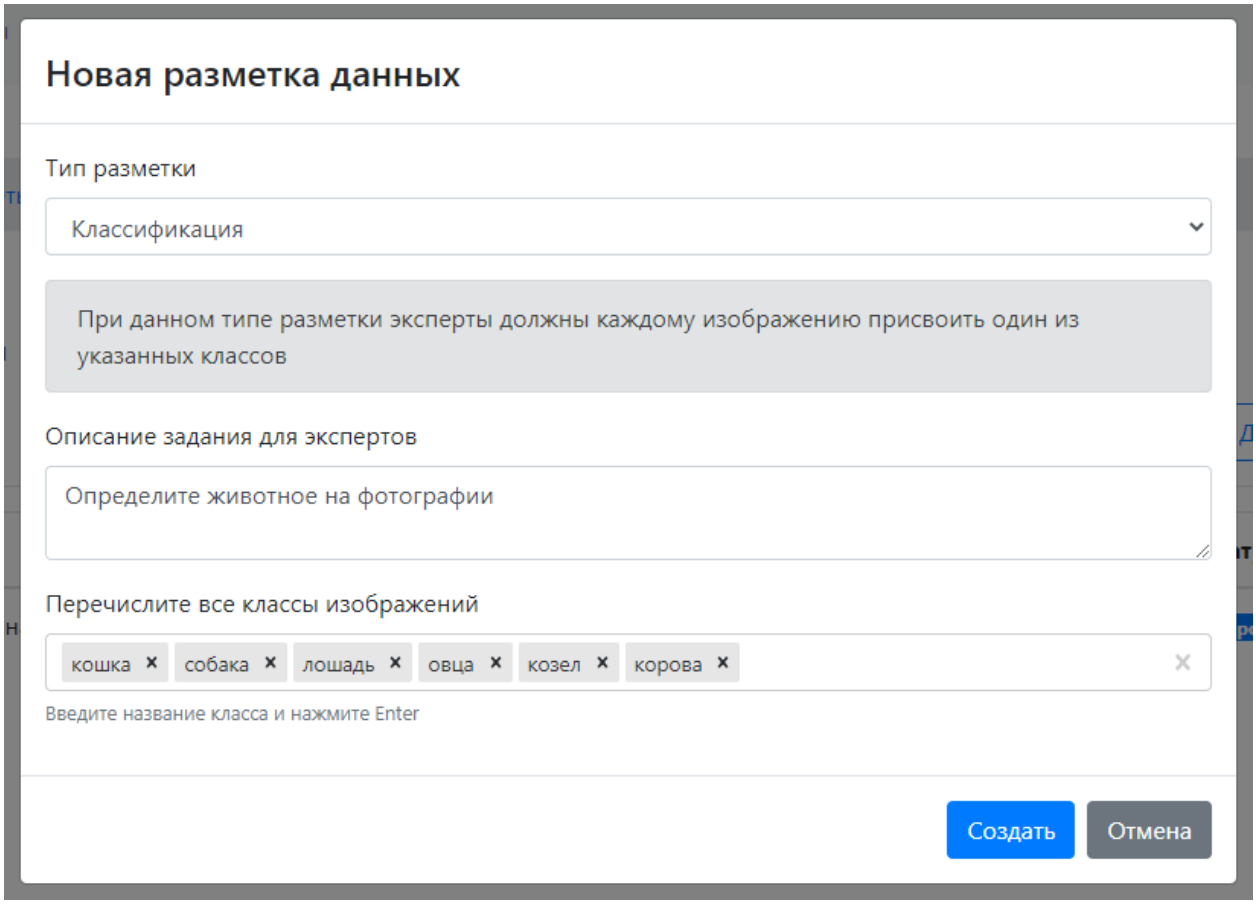
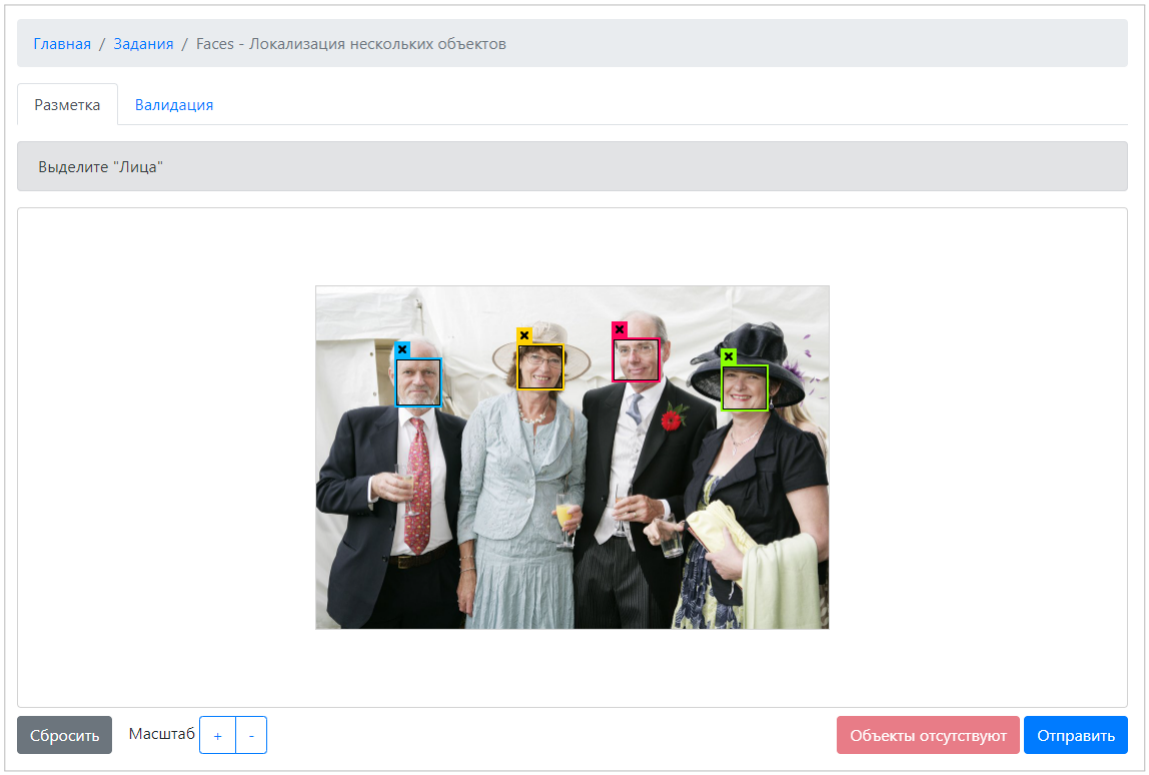
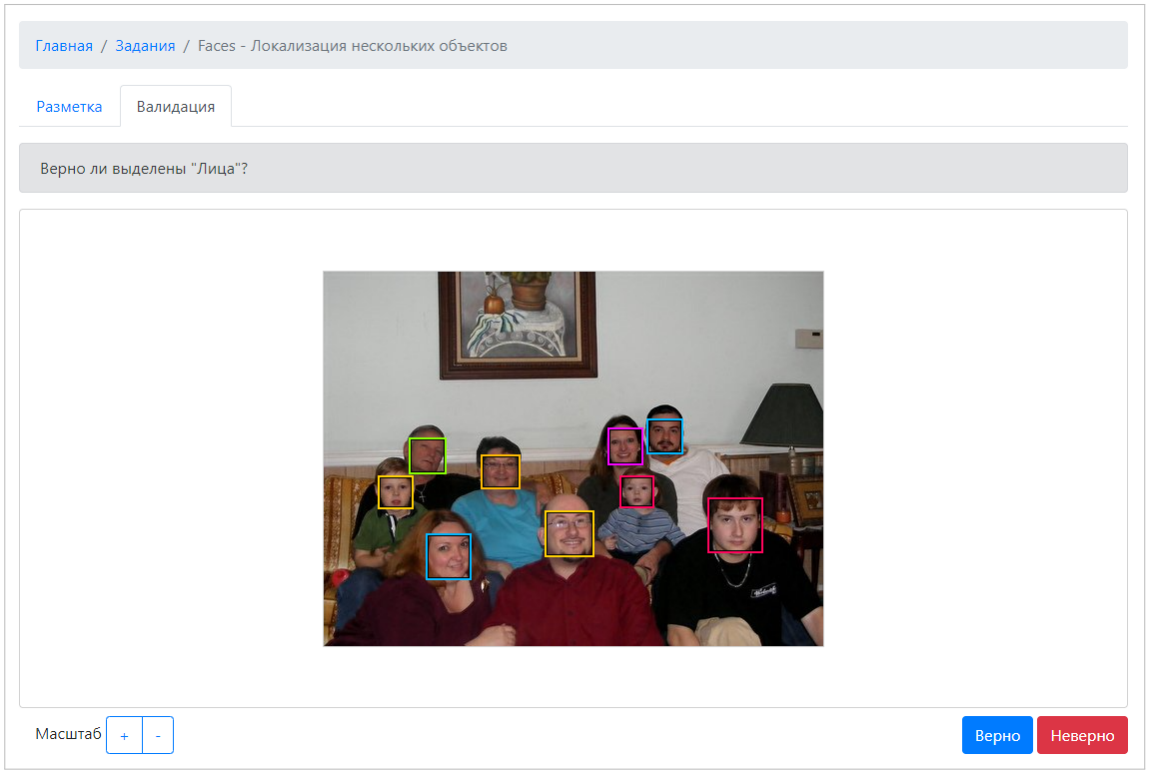


Рис. 9: Окно создания нового задания разметки.

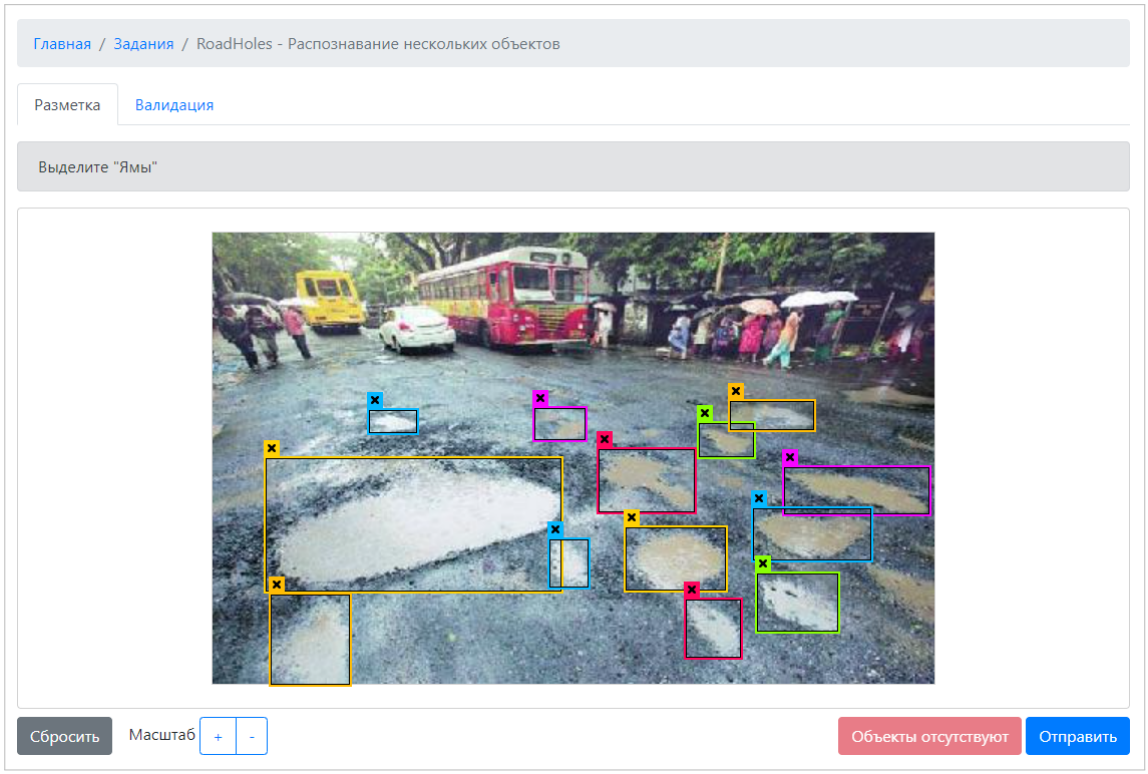


(a) Меню ручной разметки.

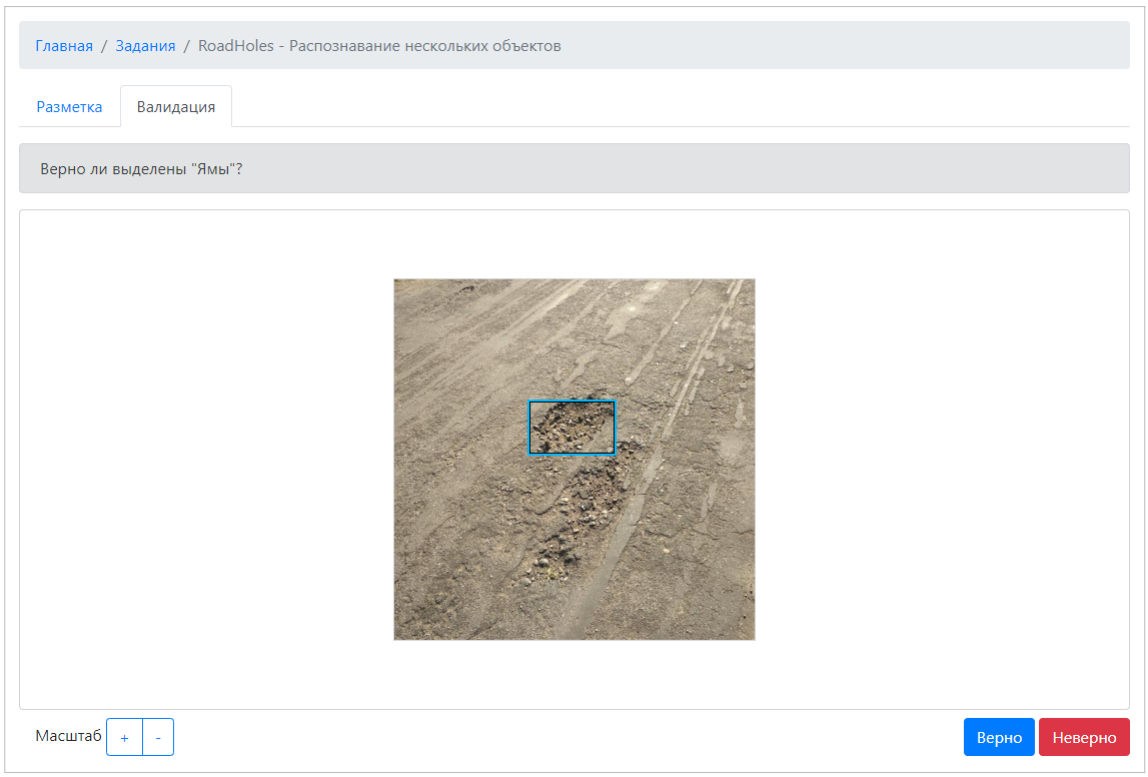


(b) Меню валидации результатов автоматической разметки

Рис. 10: Интерфейса ручной разметки (10a) и валидации автоматической разметки (10b) на примере задания разметки лиц на изображении.



(a) Меню ручной разметки



(b) Меню валидации результатов автоматической разметки

Рис. 11: Интерфейса ручной разметки (11a) и валидации автоматической разметки (11b) на примере задания разметки ям на дорогах.