

СОДЕРЖАНИЕ

1	ВВЕДЕНИЕ	3
1.1	3D-графика	3
1.2	Живые съемки	3
1.3	Автоматическая замена лиц	4
2	ПОСТАНОВКА ЗАДАЧИ	6
2.1	Метрики оценки качества замены лица	6
3	КОМАНДА РАЗРАБОТКИ	7
4	АЛЬТЕРНАТИВНЫЕ РЕШЕНИЯ	8
4.1	Инструменты	8
4.1.1	DeepFaceLab	8
4.1.2	FSGAN	9
4.2	Готовые сервисы	9
4.2.1	Reface	9
4.2.2	Другие мобильные приложения	10
5	ОПИСАНИЕ МЕТОДОВ РЕШЕНИЯ	11
5.1	Нейронная сеть	11
5.2	Инфраструктура	13
5.3	Хранение данных	14
5.4	Предобработка данных	16
6	ОСНОВНАЯ ЧАСТЬ	19
6.1	Балансировщик нагрузки	21
6.2	Взаимодействие с обработчиками	21
6.3	Клиентское приложение (фронтенд)	24
6.4	Сервер приложения (бэкенд)	24
6.5	СУБД	25

7	ЗАКЛЮЧЕНИЕ	27
7.1	Анализ полученных результатов	27
7.1.1	Оптимальная нагрузка	27
7.1.2	Перекодирование	28
7.1.3	Параллелизм	29
7.1.4	Оценка наложения лиц	29

1 ВВЕДЕНИЕ

Большая проблема при производстве видеоконтента в рекламе и других областях заключается в привязке к конкретному человеку. Используя конкретного человека, а особенно знаменитость, например, в рекламе, компании имеют дело с постоянными рисками и неудобствами. Это увеличивающиеся расходы (человек становится более известным и "растет в цене"), сложности в планировании съемок, опоздания и подобные проблемы.

1.1 3D-графика

Одним из альтернативных вариантов решения являются использование 3D-графики. В этом случае настоящие съемки практически полностью заменяются на монтаж видео с использованием 3D-моделей и дальнейшей озвучки. Для этого требуется ручной труд дизайнеров, особое техническое и программное обеспечение. В целом, данный подход дает огромные возможности, которые активно используются, например в кино, позволяя создавать контент не только с людьми, но и с любыми животными или персонажами, снимать сцены, которые невозможно снять с людьми. В то же время данный подход имеет множество недостатков, не позволяющих его использовать для производства простого и массового контента. Во-первых - трудоемкость и большие затраты во времени. Процесс создания компьютерной графики может занимать недели и месяцы и требует постоянной работы команды дизайнеров. Минимальные правки, например, одежды, снова требуют очень сложного процесса моделирования в 3D.

1.2 Живые съемки

Обычным подходом являются полностью живые съемки с живыми актерами и небольшим монтажем после. Это достаточно дешево, если не привлекать к съемкам знаменитостей, и не слишком трудоемко. Минусы данно-

го подхода следуют из привязанности к конкретному актеру. Это постоянно возвращающая стоимость в случае со знаменитостями, человеческий фактор (опоздания, личные проблемы, неподобающий внешний вид), необходимость в макияже, гриме, одежде. Так же живые съемки не позволяют создавать эффекты, возможные при использовании полноценной 3d-графики.

Несмотря на недостатки, данный подход наиболее широко используется в производстве видеоконтента. Большая часть рекламы, фильмов, видеороликов снято с живыми людьми.

1.3 Автоматическая замена лиц

С последними исследованиями в области нейронных сетей оказалось, что проблему можно решить с помощью комбинированного подхода. Современные модели нейронных сетей позволяют накладывать лицо на актера на видео автоматически, без ручного труда дизайнеров и специалистов по видеомонтажу.

Такие модели позволяют снимать видео на обычную камеру с любым актером, что гораздо быстрее и дешевле как видеомонтажа с 3d-графикой, так и полноценных съемок со знаменитостями.

Ранние решения по замене лиц требовали специального обучения модели для конкретной пары актера и лица, а так же обширный набор данных, включающий съемки головы с разных ракурсов и с разными выражениями лица. Более того, требовалась ручная доработка результата, чтобы считать его удовлетворительным. Самые последние решения дают возможность замены лица по одному снимку и обеспечивают получение результата, практически не требующего доработок.

Хотя решений в этой области достаточно много[5][14][12], все они имеют свои недостатки. Инструменты, не представленные в виде готового к использованию сервиса, сложны в применении - требуют мощностей и навыков для разворачивания всего необходимого. Существующие сервисы в основном ориентированы на массовых пользователей, а не бизнес, из-за чего имеют недостаточное качество и множество ограничений.

В данной работе рассматривается разработка альтернативного сервиса, использующего новейшие исследования в области замены лиц и предлагающего замену лиц как сервис, ориентированный на бизнес-клиентов.

2 ПОСТАНОВКА ЗАДАЧИ

Замена лица - это задача переноса лица с исходного на целевое изображение или видео так, чтобы оно аккуратно заменяло внешность лица на целевом изображении и получался реалистичный результат[12, стр. 1].

Главной задачей данной работы является разработка информационной системы, основная функция которой - автоматическая замена лиц на видео. Разработанная ИС должна быть доступна в виде публичного интернет-сервиса, которым может воспользоваться любой человек. Основной целевой аудиторией сервиса являются бизнес-клиенты, действующие в сферах маркетинга и производства видеоконтента.

Требования к разрабатываемой ИС:

- Одновременная работа множества пользователей
- Пользовательский интерфейс не требующий отдельного обучения
- Замена лица на видео по одному изображению, без необходимости в сборе большого датасета
- Хорошее качество замены лица
- Отсутствие требований к аппаратному обеспечению пользователя

2.1 Метрики оценки качества замены лица

Основные метрики оценки результата наложения лиц - качество полученного изображения и уровень похожести лиц.

Под качеством имеются в виду стандартные метрики уровня качества изображения, такие как уровень шума, чистота деталей и потеря деталей в темных областях[3], так и его реалистичность, т.е. уровень способности человеческого глаза отличить получившееся изображение от настоящей фотографии[19].

Уровень похожести двух лиц мы понимаем интуитивно как совпадение черт лица, позволяющее человеку принять два различных лица за одно и тоже[4][6][16].

3 КОМАНДА РАЗРАБОТКИ

Создать полноценный, готовый продукт такого масштаба в одиночку очень сложно. Над данным решением работала команда из нескольких человек в течение года. Команда состоит из менеджера продукта, дизайнера, двух ML-разработчиков и меня в роли разработчика клиентской и серверной части.

Часть продукта, отвечающая непосредственно за обработку видео (получает на вход видео и изображение, на выходе отдает новое видео), разрабатывалась ML-разработчиками.

Я занимался разработкой клиентского веб-приложения, серверной части, некоторых вспомогательных сервисов и оберткой вокруг нейронной сети, позволяющей встроить ее в систему.

4 АЛЬТЕРНАТИВНЫЕ РЕШЕНИЯ

4.1 Инструменты

4.1.1 DeepFaceLab

Одно из самых популярных решений в области замены лица - DeepFaceLab[14]. Оно существует с 2018 года и до сих пор остается передовой и активно развивающейся технологией.

Плюсами являются очень высокое качество наложения и хорошее сохранение похожести лица, большое количество возможностей для подстройки для конкретной пары лиц и видео, высокое разрешение лица.

Несмотря на преимущества, у DFL есть несколько серьезных недостатков, значительно усложняющих его использование. Для работы DFL требует дообучения модели для конкретной пары лиц на достаточно большом наборе данных, включающем изображения со всех ракурсов как для человека на видео, так и для накладываемого персонажа. Так же дообучение занимает значительное время и требует больших мощностей. DFL работает по принципу маски, накладывая одно лицо на другое, что делает его очень чувствительным к качеству датасетов. Так же это создает проблему разницы в освещении и цвете между видео и датасетом маски.

Упомянутые выше недостатки проявляются для потенциального пользователя в следующих минусах:

- Необходимость в сборе большого (несколько десятков или более изображений) датасета для актера и накладываемого персонажа
- После сбора датасета необходимо дообучать модель в течение нескольких недель
- Датасет должен быть очень качественным, т.е. иметь изображения с разнообразных углов и с разным освещением

4.1.2 FSGAN

FSGAN[12][13] - более современная технология, появившаяся в 2019 году.

Face reenactment (перенос мимики, puppeteering) - использование движений и мимики лица в ведущем видео для управления мимикой и движением лица, представленном на другом видео или изображении.

Алгоритм работы FSGAN состоит из следующих шагов:

- Сегментация - выделение лица на исходном и целевом изображении
- Face reenactment - воссоздание мимики и положения лица на целевом видео с исходным лицом
- Смешивание лиц (face blending) - наложение получившегося лица на целевое изображение с сохранением цвета кожи и освещения

Решение, предлагаемое FSGAN не требует дообучения для конкретных пар лиц и предлагает замену лица на основе лишь одного изображения исходного лица, против большого датасета для DeepFaceLab.

Тем не менее, FSGAN имеет довольно низкое качество наложения, так выглядит пример его работы:



Рисунок 4.1 – Примеры работы замены лица и переноса мимика в FSGAN

4.2 Готовые сервисы

4.2.1 Reface

Популярным сервисом для замены лица является Reface. Это приложение для смартфонов, позволяющее наложить свою фотографию на один из

видеороликов, предоставленных Reface. Приложение ориентировано на сегмент b2c, т.е. массового пользователя. Reface по очевидным причинам не подходит для создания видеоконтента для бизнеса. Во-первых, лица можно накладывать только на ролики из существующего набора, который содержит отрывки из популярных фильмов и сериалов. Во-вторых, выходное качество видео очень низкое - это касается как качества наложения, так и выходного разрешения, частоты кадров и других параметров видео. Сервис ориентирован на то, что роликами просто будут делиться в соцсетях и мессенджерах, а не на серьезный видеопродакшн.

Минусы Reface исходят из его позиционирования на другой сегмент, и таким образом он не является хорошей альтернативой в сегменте b2b.

4.2.2 Другие мобильные приложения

Помимо Reface похожие функции появляются во многих других мобильных приложениях. Все они ориентированы на сегмент b2c, имеют низкое качество выходного видео и множество ограничений. Большая часть таких приложений накладывает лица только на фотографии, но не видео.

5 ОПИСАНИЕ МЕТОДОВ РЕШЕНИЯ

5.1 Нейронная сеть

В процессе разработки сервиса наша команда разработала собственное решение для замены лиц на базе SimSwap[5]. SimSwap[5] появился во второй половине 2021 года и позиционируется как ”эффективный фреймворк для высокоточной замены лиц”.



Рисунок 5.1 – Результаты наложения лица в исходном SimSwap

Данное решение оказалось наиболее подходящим для наших целей. SimSwap позволяет очень качественно заменять лица, используя одну предобученную модель и всего лишь одно изображение целевого лица. При этом вместо наложения маски SimSwap использует т.н. инъекцию лицевых атрибутов. Т.е. вместо наложения одного изображения на другое модель пытается изменить лицо на изображении так, чтобы оно было похоже на другое лицо. Это позволяет избежать проблем связанных с разностью в освещении и цветовой гамме изображений. При этом модель изначально работала достаточно быстро - 1 минута видео обрабатывалась около 10-15 минут.

5.2 Инфраструктура

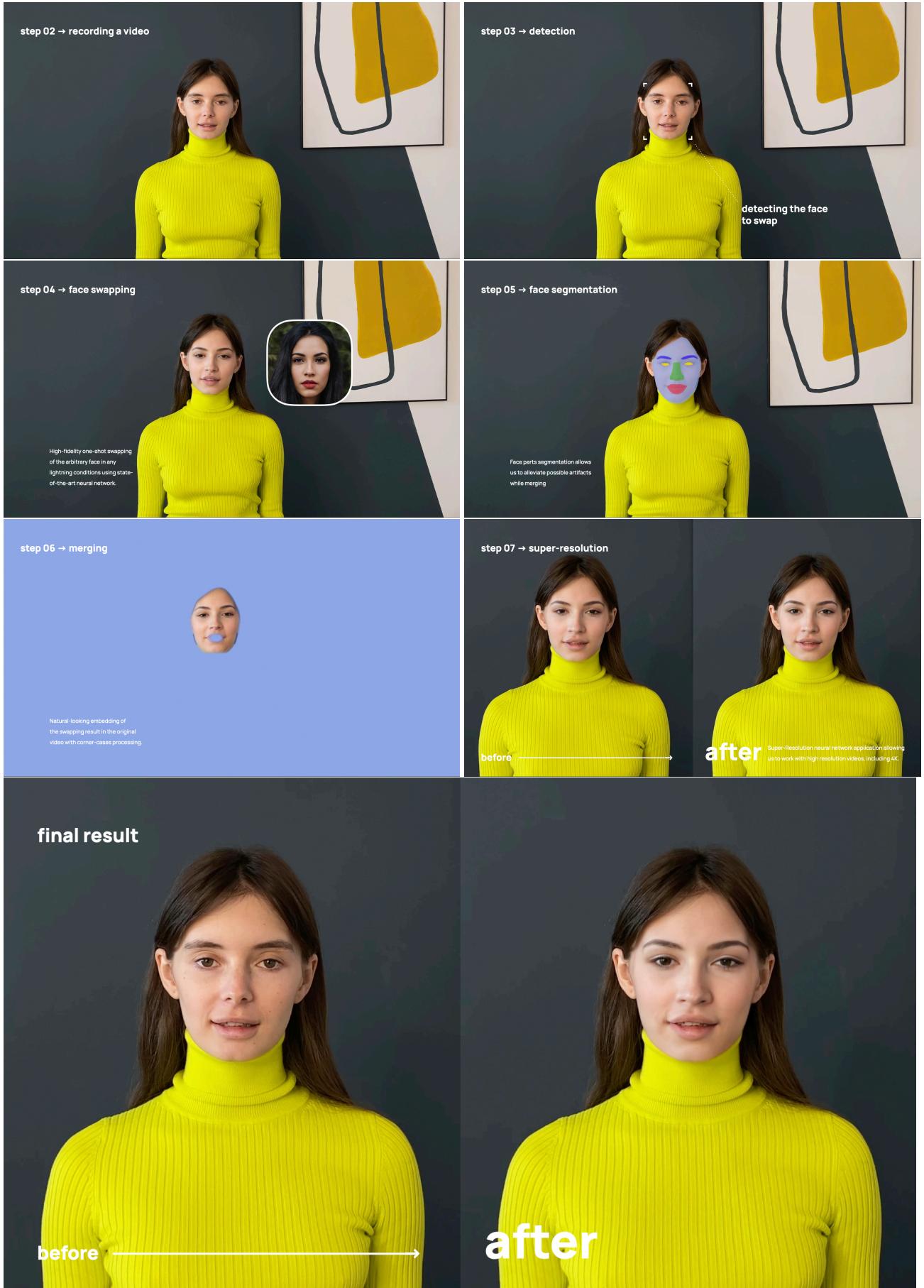


Рисунок 5.2 – Последовательность обработки видео

В любой системе центральным вопросом инфраструктуры является расположение серверов. Под разработкой обычно подразумевается создание программного продукта, но когда дело касается разворачивания приложения в публичный доступ, возникает вопрос - покупать и располагать физические сервера у себя в компании или дата-центре или воспользоваться облачными провайдерами и получать необходимые мощности по подписке.

Рассмотрим некоторые особенности обоих вариантов.

Собственные сервера приобретаются один раз и могут стоить достаточно дешево, в сравнении с ежемесячными счетами от облачного провайдера. На первый взгляд из постоянных расходов они требуют лишь электричество и интернет. Но в деталях ситуация оказывается гораздо сложнее.

Собственные сервера требуют настройки и постоянного обслуживания, пример - обновления ПО. Для того, чтобы обеспечить непрерывную работы потребуются резервные интернет-каналы, запасной источник питания (например, ИБП). Так же сервера отличаются от обычных компьютеров тем - они ориентированы на постоянную работу и имеют свои особенности. Например, шум и необходимость серьезного охлаждения. Обычно это проявляются в том, что для содержания своих серверов выделяют отдельную комнату - серверную.

Помимо собственно поддержки самого аппаратного обеспечения и операционных систем серверов, собственные сервера лишают возможности использования дополнительных услуг облачных провайдеров - управляемые (managed) сервисы, автоматическая репликация, в том числе геораспределенная, управление доступом и многое другое.

Если учитывать свою стоимость обслуживания, собственные сервера могут оказаться гораздо дороже облачных услуг.

Облачные провайдеры, напротив, предоставляют максимально удобные, готовые и самоподдерживаемые сервисы, особенно при использовании не чистых виртуальных машин, а готовых сервисов, таких как управляемые базы данных, управляемый Kubernetes, файловое хранилище. Облачные провайдеры предоставляют автоматическое резервное копирование и обновление. Серьезным минусом использования "облаков" является так называемый vendor-lock

- завязка на конкретного провайдера и невозможность (или очень высокая стоимость) переноса инфраструктуры от него. Это случается, если на проекте активно используются уникальные услуги, не предоставляемые или сильно отличающиеся у других провайдеров.

Для данного проекта было выбрано использования облачного провайдера Azure для размещения инфраструктуры. Это сервис от компании Microsoft. Он содержит огромное количество готовых сервисов и удобный веб-интерфейс для управления инфраструктурой.

Как целевая платформа для разработки приложений был выбран Kubernetes. Он имеет ряд очень существенных преимуществ по сравнению с работой с обычными серверами или виртуальными машинами.

- Позволяет автоматически разворачивать сервера и подстраивать их количество под требования системы
- Универсален для всех облачных провайдеров и позволяет легко перенести свое приложение, в том числе на собственную инфраструктуру
- Предоставляет стандартизированный способ разворачивания приложений и сервисов
- Имеет огромное сообщество, множество open-source решений имеют готовые конфигурационные файлы для Kubernetes
- Позволяет легко открывать доступ из вне к приложениям, при этом сохраняя их по умолчанию приватными
- Содержит инструменты для мониторинга и отладки приложений

5.3 Хранение данных

Исходя из специфики работы с мультимедиа в решении требуется файловое хранилище, позволяющее хранить промежуточные и финальные результаты обработки. Объектом хранения здесь являются бинарные файлы, представляющие собой фрагменты видео, целые видеофайлы, а так же фотографии лиц.

Рассмотрим различные опции:

- База данных

- Файловая система
- Оперативное хранилище, например Redis
- Облачное хранилище (S3)

Современные СУБД и частности PostgreSQL позволяют хранить бинарные данные в обычных SQL таблицах. В PostgreSQL для этого существуют 2 опции - **Large Objects** и тип данных `bytea`.

К сожалению, эти опции не очень хорошо подходят для нашего случая.

Large Objects являются нестандартной возможностью и плохо поддерживаются различными клиентами. Так же они требуют явного удаления файла, потому что сами данные хранятся отдельно от таблицы, которой принадлежат.

`bytea` плохо подходит для больших файлов, т.к. не поддерживает стриминг, а значит при работе с видеофайлами требуется много оперативной памяти. В нашем случае файлы вполне могут иметь размер в несколько гигабайт и даже больше, а значит требования к серверу базы данных (как и к клиентскому приложению) были бы огромны.

Более того - обе опции очень малопродуктивны. Это очевидно, если исходить из того, что SQL базы данных совсем не ориентированы на хранение больших бинарных блоков. Производительность запросов в этом случае будет оставлять желать лучшего.

Файловая система, что неудивительно, отлично подходит для хранения файлов. Но в нашем случае имеет ряд критических недостатков. Во-первых, к диску может одновременно иметь доступ только одна машина (виртуальная или физическая), а значит провоцирует монолитную архитектуру сервиса, что невозможно в нашем случае. Во-вторых, файловая система не позволяет автоматически переносить данные на более медленные и дешевые СХД. В-третьих, нужно самому организовывать систему резервного копирования, чтобы не потерять данные.

Третий вариант - key-value системы хранения данных, подобные Redis. Они хранят данные в памяти, иногда сгружая в постоянное хранилище. Для данного проекта они так же имеют ряд серьезных ограничений. Такие системы не гарантируют надежность хранения данных, имеют плохую устойчивость к

перезагрузкам. Они ориентированы на хранение данных в памяти, что плохо подходит для больших файлов.

Облачные файловые хранилища оказались лучшей опцией в нашем случае. Хранение большого объема файлов в нем достаточно дешево, они гарантируют надежность хранения данных и их репликацию по необходимости. Обращение к файлам являются достаточно быстрыми по скорости, особенно учитывая колокацию с серверами приложений на мощностях облачного провайдера.

Таким образом, для хранения всех медиафайлов в процессе работы проекта было выбрано облачное хранилище, а конкретно Azure Blob Storage. Решения у различных облачных провайдеров в этой области очень похожи. Для проекта самым рациональным является выбор того сервиса, который относится к используемому облачному провайдеру.

5.4 Предобработка данных

Перед наложением лица видео требует дополнительной обработки исходя из требований к сервису.

Во-первых, пользователям предлагаются разные тарифные планы, при которых обработка видео требует различного количества ресурсов. Например, для пользователей бесплатной (пробной) версии доступна обработка видео разрешением до FullHD (1920x1080) и до 30 кадров в секунду. Такие ограничения позволяют сэкономить на времени обработки, т.к. нейронные сети быстрее работают с меньшим объемом данных. Назовем этот шаг ограничением качества видео.

Во-вторых, в требованиях к сервису мы определили необходимость ускорения обработки за счет распределенной обработки на нескольких серверах одновременно.

Самый эффективный способ распределить обработку видео на несколько потоков - поделить его на сегменты примерно одинаковой длины и запустить

в очередь как обработку нескольких видео. Так сервера-обработчики смогут максимально быстро разобрать и выполнить все задачи по обработке.

В текущей версии ограничение качества не происходит на этапе предобработки данных, а вместо этого выполняется на этапе самой обработки, непосредственно перед передачей обрабатываемого сегмента в нейронную сеть.

Второй же шаг необходим. Задача состоит в том, чтобы поделить видео на сегменты примерно равной продолжительности, не потратив на это много времени. При этом после обработки эти сегменты должно быть возможно бесшовно склеить ровно так, как было в исходном видео. Еще одна задача этого этапа - максимально сохранить качество видео. Если ограничения на качество видео не накладываются, мы хотим чтобы пользователь получил результат максимально близкий к исходному видео по качеству. Так как мы ориентируемся на бизнес-пользователей, для которых это очень критично.

Так же разделение на сегменты является важной возможностью сервиса, сильно увеличивающей возможности для контроля обработки. Например, в случае неожиданной ошибки прогресс сможет быть сохранен с точностью до длины кусочка. Так же появляется возможность приоритезации обработок разных клиентов "на ходу". Во время долгой обработки бесплатного клиента, может быть загружено видео более приоритетного клиента. Тогда в случае коротких сегментов у нас будет возможность максимально быстро обработать приоритетное видео и затем вернуться к первому клиенту.

FFMpeg - утилита - швейцарский нож в мире обработки и перекодирования видео. Она содержит множество настроек и инструментов для практически любых задач в этой области. В частности, в ней есть необходимая нам возможность разделения видео на сегменты. Нужная нам опция называется /texttsegment muxer.

Для использования этой возможности нам необходимо включить в ffmpeg режим вывода segment опцией `f segment` и задать целевое время сегмента с помощью `-segment_time <количество_секунд>`. Мы используем 5 се-

кунд на сегмент. Это хорошо подходит под среднюю длину видео, загружаемых нашими клиентами, которая не превышает 10-20 секунд.

6 ОСНОВНАЯ ЧАСТЬ

Одним из требований к разрабатываемой системе является отсутствие требований к аппаратному обеспечению пользователя. Нейронная сеть для замены лица очень ресурсоемкая, для нормальной работы ей требуется видеокарта с, как минимум, 10 гигабайтами VRAM. Обработка на устройстве пользователя в таком случае сделала бы сервис неудобным или невозможным для использования для многих пользователей, в виду отсутствия полноценной видеокарты на большинстве ноутбуков, смартфонов и других устройств.

Использование устройства пользователя для вычислений так же создало бы проблемы в лицензировании и охране исходных кодов и алгоритмов, так как программу бы пришлось устанавливать на компьютер пользователя.

Данные ограничения не позволяют разработать сервис в виде desktop-приложения.

Значительно лучше в наши требования вписывается клиент-серверная модель. В ней мы можем вынести большую часть системы на сервер, скрывая исполняемый код от пользователя и производя вычисления на собственной инфраструктуре. Для пользователя в таком случае необходимо предоставить лишь "тонкий" клиент, который может работать на любой платформе, в т.ч. на смартфоне или планшете.

В требованиях к системе присутствует "одновременная работа множества пользователей". В зависимости от количества пользователей одновременная их работа в клиент-серверной может потребовать значительных вычислительных мощностей. Соответственно с ростом количества пользователей системы возникает необходимость в увеличении доступных ресурсов. Этот процесс и способность к нему системы называется масштабированием.

Различают горизонтальное и вертикальное масштабирование. Вертикальное - это простая замена сервера на более мощный. Например, если сервер с 8 процессорными ядрами и 32 гигабайтами памяти перестает справляться с нагрузками, его можно просто заменить на более мощный с, например, 16

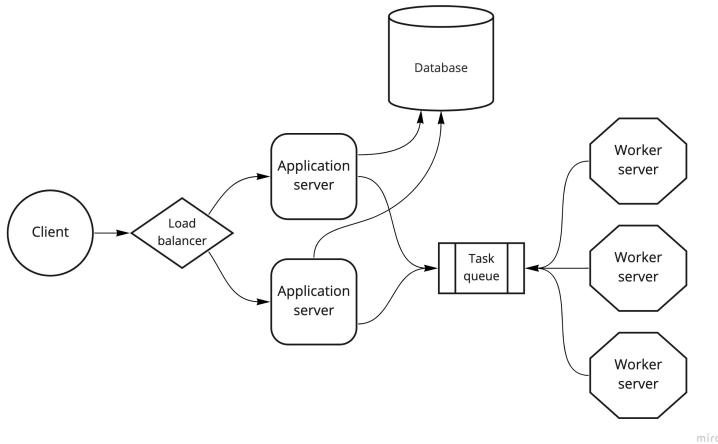


Рисунок 6.1 – Четырехзвенная архитектура

ядрами и 64 гигабайтами памяти. Более гибким считается горизонтальное масштабирование. Вместо увеличения мощности одного узла, оно подразумевает запуск дополнительных узлов, выполняющих одну и ту же роль в системе. В большинстве случаев это требует от масштабируемого компонента отсутствие собственного состояния, которое иначе создает проблему рассинхронизации данных при параллельной работе нескольких экземпляров приложения.

Исходя из необходимости поддержки масштабирования для множества пользователей и ресурсоемкости системы, при разработке изначально было принято решение в необходимости поддержки горизонтального масштабирования системы.

Наиболее важная и ресурсозатратная функция разрабатываемой системы - замена лица. Для поддержки одновременной работы множества пользователей нам необходимо иметь возможность легко масштабировать компонент, выполняющий эту функцию.

Данные размышления приводят нам к архитектуре, состоящей из 4 звеньев - клиент, сервер, база данных и обработчик.

Данная архитектура позволяет легко дублировать и масштабировать наиболее важные компоненты - сервер приложения и сервера обработчики. При этом в местах соединения одного компонента со многими возникает необходимость в дополнительных узлах, выполняющих функцию распределения нагрузки и распределения соединений с конкретным узлом.

6.1 Балансировщик нагрузки

При наличии нескольких серверов приложений требуется балансировщик нагрузки (load balancer) - это дополнительный сервер, который получает запрос от клиента и перенаправляет его в один из целевых серверов приложений. Балансировщик нагрузки равномерно распределяет нагрузку по серверам, а также позволяет при недоступности одного или нескольких серверов, направлять подключения на те, которые готовы обрабатывать запросы. Балансировщик работает с синхронными запросами по таким протоколам, как HTTP или GRPC. Microsoft Azure, как и другие облачные провайдеры, предоставляет возможность арендовать готовый к использованию балансировщик.

6.2 Взаимодействие с обработчиками

При соединении серверов приложений с серверами обработчиками возникает необходимость во взаимодействии другого вида. При использовании обычного балансировщика нагрузки возникла бы необходимость держать соединение между этими серверами все время обработки запроса, который может занимать до нескольких часов. К тому же балансировщик не позволяет "откладывать" соединения на потом, когда свободных серверов для обработки нет.

Для этого соединения я выбрал распределенную очередь задач. В данной случае вместо балансировщика нагрузки используется брокер сообщений, позволяющий сохранять задачи, требующие обработки, в очереди. Одно из наиболее готовых к использованию "из коробки" решений в этой области - библиотека Celery для Python[17]. Она реализует всю необходимую логику для описания задач на языке программирования, содержит протокол передачи задач и данных через брокер сообщений, а так же имеет адаптеры к множеству брокеров, таких как Redis, RabbitMQ или Amazon SQS.

На роль брокера сообщений был выбран RabbitMQ, который в отличие от Redis имеет более полные гарантии сохранности данных, и в отличие от Amazon

SQS является решением с открытым исходным кодом, которое можно запустить на собственной инфраструктуре, не привязываясь к конкретному облачному провайдеру.

Минимальный пример использования Celery с RabbitMQ на языке Python может выглядеть так:

Описываем приложение Celery с функцией сложения двух чисел, помеченной декоратором `@app.task`:

```
from celery import Celery

app = Celery('tasks', broker='amqp://guest@localhost//')

@app.task
def add(x, y):
    return x + y
```

Запустить сервер обработчик Celery можно так

`celery -A tasks worker.`

Вызов функции и получение результата:

```
from tasks import add

result = add.delay(4, 4)
print(result.get(timeout=1))
```

Celery так же предоставляет возможность различным образом комбинировать задачи, составляя цепочки выполнения заданий. В рамках этой возможности существует несколько примитивов, используя которые можно достичь необходимого результата. Разберем несколько таких примитивов:

- `group` - позволяет выполнить несколько задач параллельно, получив одновременно все результаты
- `chain` - позволяет связать выполнение нескольких последовательных задач, передавая результаты выполнения по цепочке
- `chord` - комбинирует `group` и задачу, получающую на вход результаты выполнения группы задач

```

@app.task(
    # игнорируем возвращаемое значение , тк..
    # результат записывается в БД внутри
    ignore_result=True,
    # используем базовый класс задачи для обработки ошибок
    base=processing.FaceSwapTaskBase,
    # гарантирует что при поломке обработчика задача будет сохранена
    acks_late=True,
    # через 3 часа будет брошено исключение SoftTimeLimitExceeded
    soft_time_limit=10800,
    # еще через 200 секунд принудительно завершает выполнение
    time_limit=11000,
    # пытаемся выполнить задачу заного , если брошено исключение
    # SoftTimeLimitExceeded
    autoretry_for=(SoftTimeLimitExceeded,),
    # максимальное делаем 3 попытки обработки задачи
    retry_kwargs={"max_retries": 3},
)
@as_sync
async def execute_face_swap_request(request_id: int): #
    на вход получаем id задачи из БД
    # выполняем препроцессинг и делим видео на сегменты
    segment_tasks = await processing.preprocess_face_swap(
        request_id)
    signatures = [
        # для каждого сегмента видео :
        # используем примитив chain
        # для связки задачи замены лица из задачи по обработке окончания сегмента
        # при ошибке создаем задачу handle_swap_error
        chain(face_swap.s(asdict(segment)), handle_finished_event.s
              ()).on_error(
            handle_swap_error.si(request_id)
        )
        for segment in segment_tasks
    ]
    # запускаем все задачи по обработке сегментов параллельно
    group(*signatures)()

```

6.3 Клиентское приложение (фронтенд)

Одно из преимуществ клиент-серверных архитектур - возможность независимой реализации нескольких клиентов к одному серверу. Это позволяет иметь отдельные приложения для различных платформ, таких как ПК, iOS, Android или WEB. Последняя (WEB) выделяется из этого ряда тем, что может работать на всех остальных. Так, любой сайт можно открыть практически на любом доступном устройстве. Поэтому веб является максимально универсальной платформой и избавляет от необходимости разработки других клиентов. Исходя из этого, в качестве первого (и пока единственного) клиентского приложения для реализации был выбран именно WEB.

На данный момент существует множество подходов и технологий реализации веб-приложений, рассмотрение каждого из которых невозможно в рамках данной работы. Мною для разработки был выбран язык Typescript и фреймворк SvelteKit[9]. Это одна из наиболее передовых комбинаций технологий разработки веб-приложений, из плюсов которой можно отметить поддерживаемость, скорость работы и удобство разработки.

Клиент обращается к серверу по протоколу HTTP, используя JSON в качестве формата сообщений.

6.4 Сервер приложения (бэкенд)

Основная задача бэкенда в нашей архитектуре - реализация логики работы с различными сущностями и компонентами системы и предоставления внешнего API для клиентов. В качестве архитектурного стиля для реализации API выбран REST[11] на базе JSON и HTTP в качестве протокола передачи данных.

Для создания API была выбрана библиотека FastAPI [15]. Для работы с базой данных - библиотека SQLAlchemy и Alembic в роли движка миграций.

6.5 СУБД

СУБД используется для хранения списка пользователей и их атрибутов, истории запросов на замену лица и данных о состоянии обработки этих запросов, информации о подписках и правах пользователей и многих других данных.

Основным выбором в области СУБД является использование SQL или NoSQL базы данных. SQL совместимые БД являются де-факто стандартом в области хранения данных. Их особенностью является использования реляционной модели данных[7]. NoSQL появились в ответ на необходимость в горизонтальном масштабировании баз данных при хранении огромных объемов данных, которое сильно ограничено при использовании SQL. [18]

Так, основными аргументами для использования NoSQL решений является работа с большими данными и обеспечение высокой доступности при отключении одного из узлов. В данной работе эти проблемы не являются существенными. Так, например, после полугода эксплуатации сервиса в боевом режиме объем хранимых в СУБД данных не превышает 1ГБ, а высокая доступность сервиса ограничивается по большей части другими факторами и не является критичной для бизнеса.

SQL же базы данные предоставляют более существенные для данного проекта преимущества, такие как ACID-транзакции[8], совместимость и простота миграции между СУБД, типизация моделей данных[2].

В итоге, на роль СУБД в разрабатываемом сервиса была выбрана PostgreSQL. От других подобных решений она отличается полной бесплатностью и открытым исходным кодом, огромным сообществом, множеством возможностей и дополнений, работой "из коробки" с практически любыми языками и библиотеками.

В пользу PostgreSQL так же играет возможность аренды преднастроенного и полностью управляемого сервера БД у большинства облачных провайдеров, в том числе Microsoft Azure[1]. Облачные провайдеры гарантируют определенный уровень доступности сервера, предоставляют автоматическое резервное копирование и обновление сервера, а так же легкое вертикальное масштабирование и мониторинг.

При работе с БД было решено использовать ORM [10] фреймворк SQLAlchemy и систему миграций Alembic для ЯП Python. Это позволяет отказаться от написания запросов на чистом SQL, таким образом избежав привязки к конкретномуialectу SQL и упростить возможную миграцию на другую СУБД.

7 ЗАКЛЮЧЕНИЕ

В результате данной работы была разработана система, удовлетворяющая всем требованиям обозначенным при постановке задачи.

Система поддерживает работу множества пользователей, используя веб-интерфейс все пользователи могут одновременно создавать заявки для замены лица, при это заявку могут исполняться несколькими серверами-обработчиками одновременно в порядке очереди.

Разработанный пользовательский интерфейс в достаточной степени интуитивно понятен и позволяет пользоваться сервисом без предварительной подготовки и обучения.

Замена производится лишь по одному изображению исходного лица, загружаемого или сгенерированного, а затем сохраненного в сервисе.

Качество замены лица находится на достаточно высоком уровне, хотя и имеет некоторые проблемы, описанные более подробно ниже в тексте.

При этом сервисом может воспользоваться любой пользователь, имеющий электронное устройство с доступом в интернет. Никаких особых требований к программному и аппаратному обеспечению пользователя нет.

7.1 Анализ полученных результатов

7.1.1 Оптимальная нагрузка

В процессе обработки видео задача распределяется по всем доступным серверам-обработчикам, обеспечивая максимально возможную скорость обработки.

Как только освобождается один из обработчиков, он сразу берет на себя следующую задачу для обработки, максимально используя вычислительные мощности для удовлетворения потребностей в обработке.

На данный момент не решена задача приоритезации задач для определенных классов пользователей (например, для крупных клиентов), все задачи приоритезированы одинаково и выполняются максимально быстро.

Проблема такого подхода состоит, например, в том, что платящие клиенты теоретически могут оказаться в очереди за бесплатными клиентами, что не является оптимальным с точки зрения бизнеса.

В планах на будущее присутствует возможность создания нескольких очередей или очереди с приоритетами для решения этой проблемы.

7.1.2 Перекодирование

Видео перекодируется несколько раз в ходе обработки, что увеличивает потери в качестве, происходит потеря информации из исходного видео. Так же это замедляет обработку из-за дополнительных операций кодирования/декодирования.

Более оптимальным с точки зрения минимизации потерь качества является передача медиа внутри системы в несжатом виде. Но этот подход связан с очень высокой нагрузкой на IO (системы ввода-вывода), в частности дисковые хранилища и сеть, а значит заставляет использовать более объемные и быстрые хранилища для серверов и беспокоится об объеме сетевого трафика внутри кластера. Например, 10 секунд несжатого 4K (3840x2160) видео с 60 кадрами в секунду обычного качества (SDR с форматом пикселей yuv420p) занимает 7.5 ГБ дискового пространства. При скорости соединения между узлами в 1 Гбит/с (и соответствующей скорости записи/чтения с диска в 125 МБ/с) передача такого файла займет минимум 1 минуту, что при необходимости передач через несколько узлов может занимать значительное время относительно общего времени обработки.

Поэтому данный подход не может быть использован повсеместно и имеет свои ограничения в применимости.

На данный момент мы изучаем возможность применения передачи несжатых медиа.

7.1.3 Параллелизм

Параллельная обработка на нескольких серверах позволяет максимально быстро и оптимально с точки зрения нагрузки выполнить определенный объем работы. Но с точки зрения множества пользователей это не всегда хорошо. Отработка достаточно длинного файла занимает работой сразу все сервера-обработчики. Если в этот момент на обработку поступает видео другого пользователя, особенно если оно очень короткое, происходит затор. Пользователь с длинным видео готов подождать, но будет сильно задерживать пользователя, например, с картинкой. По сути параллельная обработка заставляет использовать всю пропускную способность сервиса сразу, не оставляя места для параллельной работы с разными клиентами. Так параллельная с точки зрения нагрузки обработка становится последовательной с точки зрения множества пользователей.

Эта проблема решается определенными настройками, позволяющими балансировать между двумя подходами. Например, параллельную обработку для одного пользователя можно ограничить двумя или тремя потоками. Мы потеряем в максимальной скорости, но при этом оставим место другим обработкам в очереди, даже при обработке очень длинного видеофайла. Алгоритм для работы таких ограничений пока не реализован.

7.1.4 Оценка наложения лиц

Наша команда постоянно работает над улучшением метрик качества наложения лиц. Тем не менее при нахождении лиц вблизи качество изображения значительно ухудшается в связи с ограниченным разрешением, на котором работает нейронная сеть изменяющая лицо. Алгоритм super-resolution несколько улучшает эту ситуацию, но не исправляет окончательно.

Так же уровень сохранения похожести в разработанной системе недостаточен при использовании сильно различающихся лиц для наложения. Получившееся лицо является неким "средним" между исходным и целевым лицом.

7 СПИСОК ЛИТЕРАТУРЫ

1. Microsoft Azure. Azure database for postgresql. URL: <https://azure.microsoft.com/en-us/services/postgresql/>.
2. IBM Cloud Databases Benjamin Anderson, STSM and IBM Cloud Databases Brad Nicholson, Senior Database Engineer. Sql vs. nosql databases: What's the difference?, 2021. URL: <https://www.ibm.com/cloud/blog/sql-vs-nosql>.
3. Norman Burningham, Zygmunt Pizlo, and Jan P Allebach. Image quality metrics. *Encyclopedia of imaging science and technology*, 1:598–616, 2002.
4. Qiong Cao, Yiming Ying, and Peng Li. Similarity metric learning for face recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 2408–2415, 2013.
5. Renwang Chen, Xuanhong Chen, Bingbing Ni, and Yanhao Ge. Simswap: An efficient framework for high fidelity face swapping. In *MM '20: The 28th ACM International Conference on Multimedia*, 2020.
6. Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.
7. Chris J Date and Colin J White. *A guide to DB2*. Addison-Wesley Longman Publishing Co., Inc., 1988.
8. Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM computing surveys (CSUR)*, 15(4):287–317, 1983.
9. Rich Harris. Sveltekit - cybernetically enhanced web-apps, 2021-2022. URL: <https://kit.svelte.dev/docs/introduction#what-is-sveltekit>.

10. Mike Keith and Merrick Schnicariol. Object-relational mapping. In *Pro JPA 2*, pages 69–106. Springer, 2009.
11. Mark Masse. *REST API design rulebook: designing consistent RESTful web service interfaces.* ” O'Reilly Media, Inc.”, 2011.
12. Yuval Nirkin, Yosi Keller, and Tal Hassner. FSGAN: Subject agnostic face swapping and reenactment. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7184–7193, 2019.
13. Yuval Nirkin, Yosi Keller, and Tal Hassner. FSGANv2: Improved subject agnostic face swapping and reenactment. IEEE, 2022.
14. Ivan Perov, Daiheng Gao, Nikolay Chervonyi, Kunlin Liu, Sugasa Marangonda, Chris Umé, Mr. Dpfks, Carl Shift Facenheim, Luis RP, Jian Jiang, Sheng Zhang, Pingyu Wu, Bo Zhou, and Weiming Zhang. Deepfacelab: Integrated, flexible and extensible face-swapping framework, 2020. URL: <https://arxiv.org/abs/2005.05535>, doi:10.48550/ARXIV.2005.05535.
15. Sebastian Ramirez. Fastapi documentation. URL: <https://fastapi.tiangolo.com>.
16. Amir Sadovnik, Wassim Gharbi, Thanh Vu, and Andrew Gallagher. Finding your lookalike: Measuring face similarity rather than face identity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2345–2353, 2018.
17. Ask Solem and contributors. Celery documentation, 2009-2021. URL: <https://docs.celeryq.dev/en/stable/index.html>.
18. Christof Strauch, Ultra-Large Scale Sites, and Walter Kriha. Nosql databases. *Lecture Notes, Stuttgart Media University*, 20:24, 2011.
19. Su Xue, Aseem Agarwala, Julie Dorsey, and Holly Rushmeier. Understanding and improving the realism of image composites. *ACM Transactions on graphics (TOG)*, 31(4):1–10, 2012.