Московский авиационный институт (национальный исследовательский университет)
Институт №8 «Компьютерные науки и прикладная математика»

Проект на тему:

«Telegram бот стилизации изображений»

Исполнители:
Журавлев Кирилл Игоревич
Минеева Светлана Алексеевна
Русаков Александр Владиславович

Москва - 2025

Цели проекта

- **ML цель:** достижение высокой точности в задаче стилизации изображений и фильтрации неподобающего контента, обеспечив при этом быструю и качественную обработку запросов пользователей.

- **Бизнес цель:** создание инновационного платного сервиса для стилизации изображений в Telegram, который предлагает пользователям уникальные художественные стили на основе мощной нейронной сети. Основная цель — монетизация через модель подписки и платные запросы, что позволит пользователям получать доступ к высококачественным результатам стилизации.

Описание проекта

Описание проекта: Мы разработали Telegram bot, который умеет стилизовать полученные изображения от пользователя с помощью сверточной нейронной сети VGG16, обученной на пяти стилях. Предварительно изображение проверяется на наличие негативного контента с помощью использования готовой модели NSFW Classifier. Телеграм бот реализован с помощью библиотеки aiogram в асинхронном режиме. Для хранения запросов пользователей в виде «ключ-значение» используется хранилище данных Redis. Очереди сообщений реализуются с помощью брокера сообщений RabbitMQ. Загрузка обученных весов моделей и хранение присылаемых пользователем изображений и сгенерированных изображений реализуются с помощью облачного хранилища S3 Evolution Object Storage.

Исполнители проекта:

ФИО	Группа	Обязанности
Журавлев Кирилл Игоревич	М8О-410Б-21	анализ литературы, настройка работы инфраструктуры cloud.ru, разработка Telegram бота,настройка взаимодействия сервера и бота с облачным хранилищем, оформление документации, создание презентации
Минеева Светлана Алексеевна	М8О-410Б-21	обучение модели стилизации, интеграция моделей стилизации и детекции, разработка Telegram-бота и его интеграция с сервером, реализация внутренней логики сервера, оформление документации
Русаков Александр Владиславович	М8О-410Б-21	развертывание виртуальной машины, реализация внутренней логики сервера,настройка взаимодействия сервера и бота с облачным хранилищем, дебаггинг кода, оформление документации, съемка видео-демонстрации работы проекта

ML обучение стилизации

Стилизация изображения – это задача машинного обучения, в которой требуется преобразовать входное изображение таким образом, чтобы сохранить его содержательное представление, но изменить визуальный стиль, основанный на референсном изображении.

Определим цель МО: оптимизация взвешенной функции потерь, где учитываются 2 основные компоненты.

1. **Потери содержания (Content Loss)**

Определяется как среднеквадратичная ошибка (MSE) между функциями, извлеченными из предобученной VGG16 для оригинального изображения и стилизованного изображения. Потери содержания сохраняют структуру и объекты исходного изображения.

$$\mathcal{L}_{content} = \lambda_{content} \cdot \text{MSE}(f^{relu2\_2}_{original}, f^{relu2\_2}_{transformed})$$

где:

- $f^{relu2\_2}_{original}$ и $f^{relu2\_2}_{transformed}$ — признаки, извлеченные на слое ReLU2_2 VGG16.

2. **Потери стиля (Style Loss)**

Определяется как ошибка между матрицами Грама (описание корреляций между признаками) для стиля и сгенерированного изображения. Потери стиля обеспечивают соответствие стилевым характеристикам изображения.

$$\mathcal{L}_{style} = \lambda_{style} \cdot \sum_{I} \text{MSE}(G^{I}_{transformed}, G^{I}_{style})$$

где:

- $G^{I}_{transformed}$ и G^{I}_{style} — матрицы Грама для уровня I.

ML обучение стилизации

3. Общая функция потерь

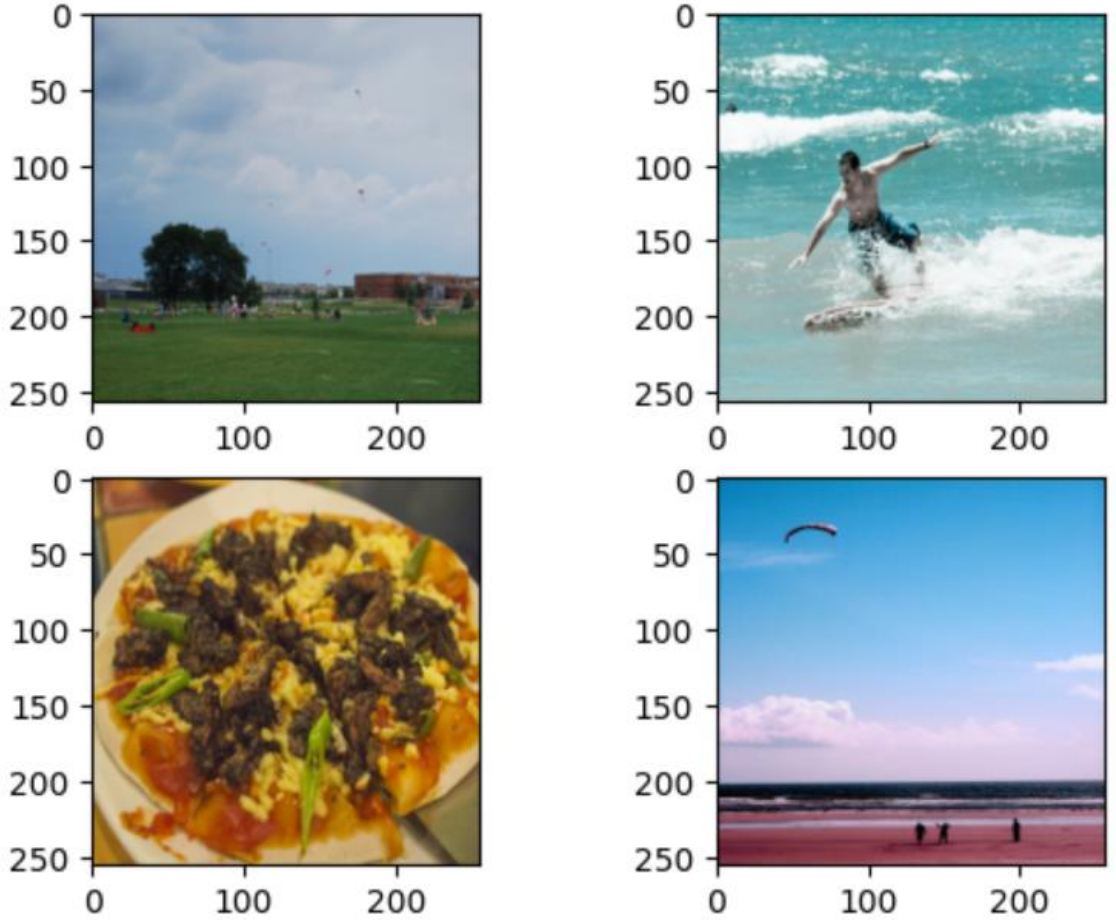
Общая функция потерь объединяет две вышеописанные компоненты с использованием весов:

$$\mathcal{L}_{total} = \mathcal{L}_{content} + \mathcal{L}_{style}$$

Мы выбрали подход взвешенной оптимизации, где баланс между содержанием и стилем достигается за счет параметров $\lambda_{content}$ и λ_{style}, поскольку такой подход позволяет достигнуть наиболее высокой точности в стилизации изображений.

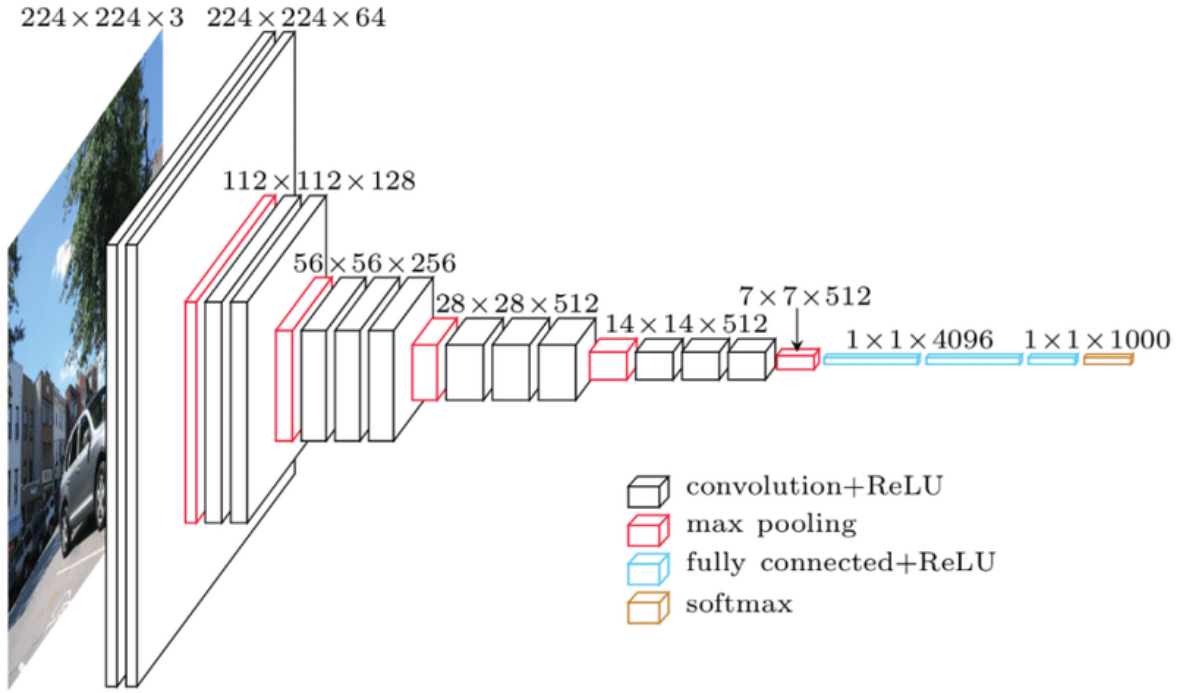
ML обучение стилизации

Dataset: Для обучения модели используется датасет - COCO2014, обучающая выборка которого состоит из 82784 изображений на произвольную тему. Ниже представлены примеры исходных данных



ML обучение стилизации

Выбор модели: Чтобы получить представление контента и стиля изображения, нужно посмотреть на промежуточные слои модели. Промежуточные слои представляют собой карты признаков, которые по мере углубления становятся более упорядоченными. В этом случае будем использовать сетевую архитектуру VGG16 - предварительно подготовленную свёрточную нейронную сеть выделения признаков изображений. Данная архитектура является отличным строительным блоком для обучения её легко реализовать, однако она имеет и несколько недостатков: медленная скорость обучения, большой вес самой архитектуры. Ниже приведена структура VGG16.



ML обучение стилизации

Для обучения модели стилизации изображения используются признаки (фичи), извлекаемые из предобученной модели **VGG16**

Примеры признаков:

- Признаки содержания извлекаются из слоя ReLU2_2 для сохранения структуры изображения, поскольку этот слой располагается в ранней части модели и сохраняет больше информации о низкоуровневой структуре изображения (края, текстуры)
- Признаки стиля берутся из слоев: ReLU1_1, ReLU2_1, ReLU3_1 и ReLU4_1 для передачи визуальных характеристик стиля, поскольку слои в разных частях сети кодируют признаки на разных уровнях абстракции. Ранние слои (ReLU1_1, ReLU2_1) кодируют текстуры и цветовые паттерны, а поздние слои (ReLU3_1 и ReLU4_1) – сложные текстуры и глобальные зависимости
- Матрица Грама позволяет представить стиль изображения в виде зависимостей между каналами, игнорируя пространственное расположение

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}$$

где:

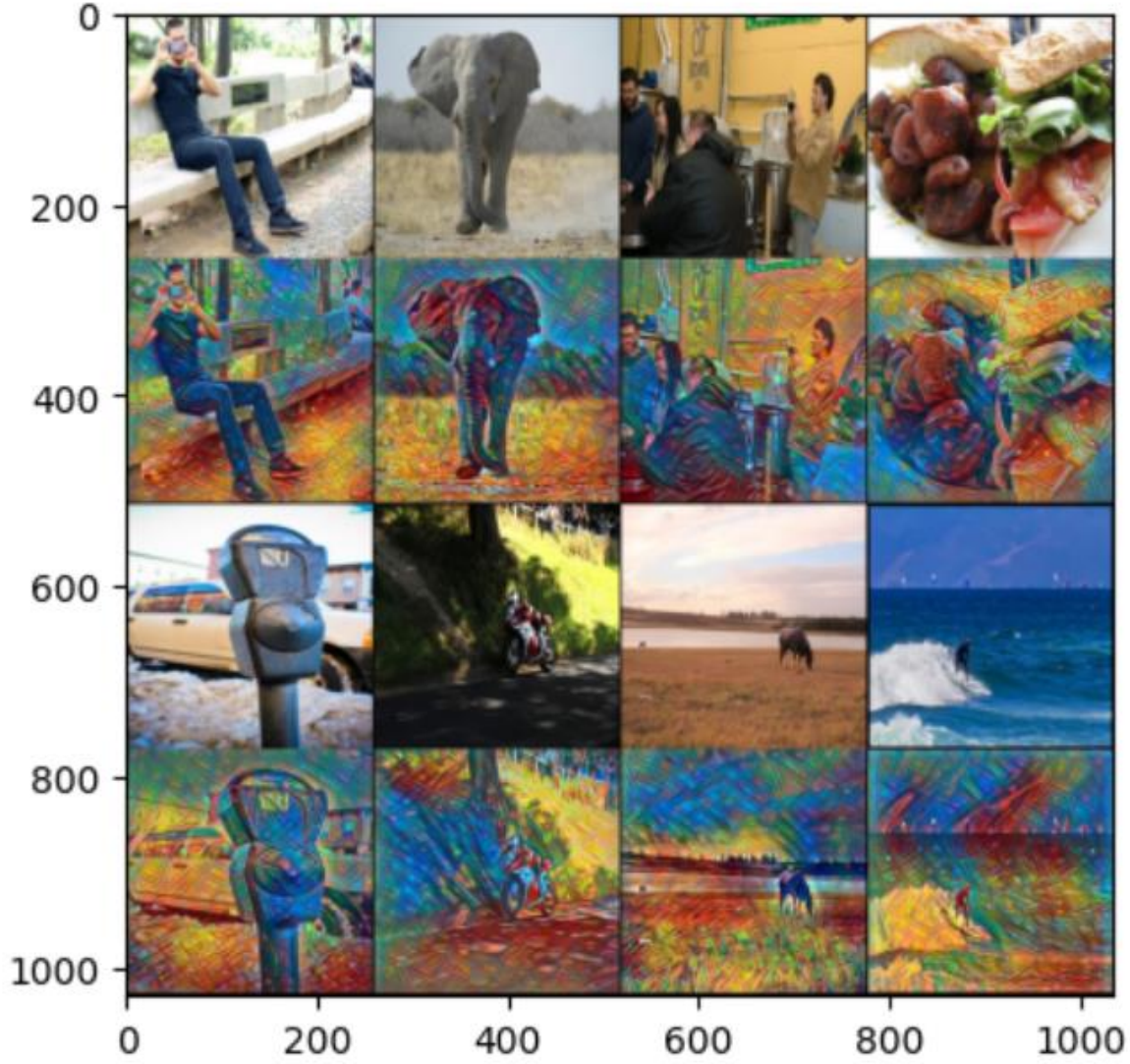
- F_{ik}^{l} — активации признаков i-го и j-го каналов на уровне l.
- k — размерность пространственной карты признаков.

ML обучение стилизации

Теперь покажем как обучается модель стилизации. Модель обучалась на пяти различных произведениях искусств с интересными стилями. Обучение каждому стилю состояло из 10000 итераций. Посмотрим на несколько стилевых изображений, с остальными можно ознакомиться в репозитории:

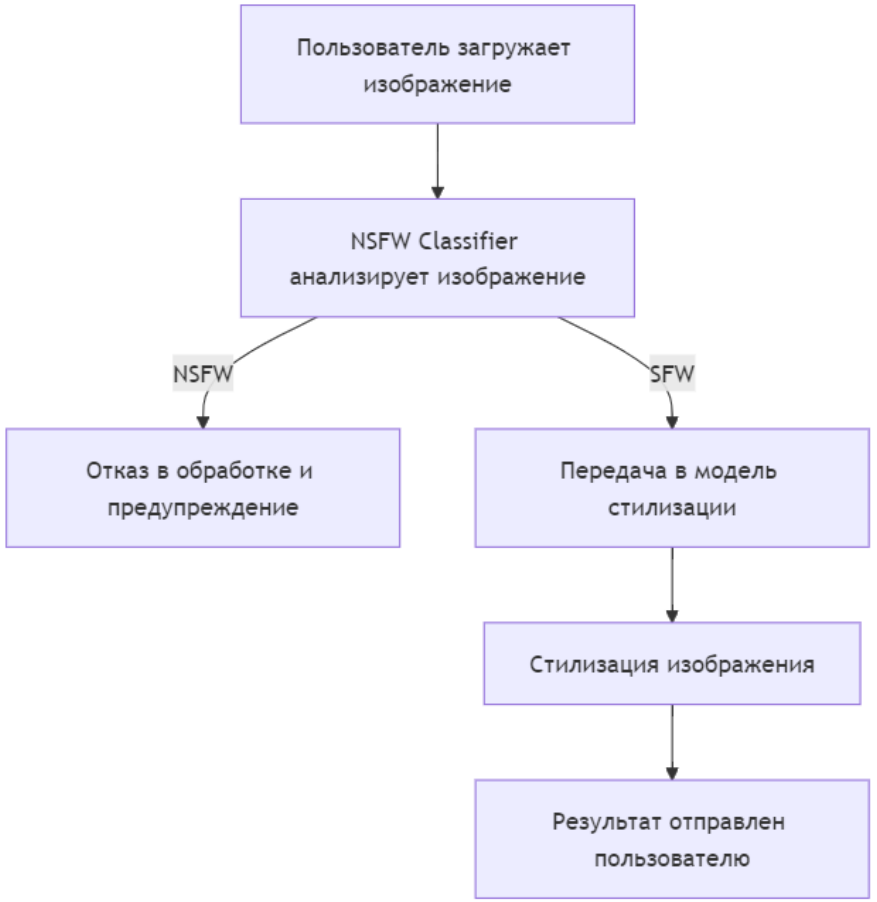
ML обучение стилизации

На слайде можно увидеть примеры обучения модели. Результатами обучения мы остались довольны. От стилевого изображения очень точно берется цветовая гамма, а также достаточно качественно - текстура картины. От контентого изображения чётко сохраняются объекты, часто даже до мельчайших подробностей. Обучение модели прошло успешно.



ML модель детекции

Использование NSFW Classifier перед моделью стилизации изображений необходимо для предотвращения стилизации неподобающего контента, который может быть неэтичным, незаконным. Ниже представлен граф, то как встроена эта модель в наш проект



Ввод ML модели в эксплуатацию

Введение модели стилизации изображений и классификатора неподобающего контента (NSFW-классификатора) в эксплуатацию предполагает, что эти модели имеют статический характер и не требуют регулярного обновления или переобучения.

Причины отсутствия необходимости в переобучении модели стилизации:

1. Завершённое обучение:
Модель обучается под конкретный набор стилей. После завершения процесса обучения она способна применять указанные стили к любым входным изображениям. Нет необходимости в динамическом обновлении модели, так как набор стилей фиксирован.

2. Статичность задачи:
Целевая задача модели — стилизация изображений — не изменяется со временем. Модель будет продолжать работать стабильно при тех же входных данных.

3. Отсутствие зависимости от новых данных:
Обучение модели не зависит от реального времени или потока новых данных. Пользователи лишь предоставляют контентные изображения, которые не требуют обновления обучающей выборки.

4. Простота добавления новых стилей:
Если требуется добавление нового стиля, можно просто обучить новую модель с другим стилевым изображением, не затрагивая текущую версию модели.

Ввод ML модели в эксплуатацию

Причины отсутствия необходимости в переобучении модели распознавания неподобающего контента :

1. Стабильные категории классификации:
Классы «NSFW» и «SFW» остаются неизменными со временем. Нет необходимости переобучать модель, если категорий или требований не становится больше.

2. Высокий объём данных на этапе обучения:
Модель обучена на обширном наборе данных, который охватывает множество сценариев, включая разные типы контента. Это позволяет ей быть достаточно универсальной.

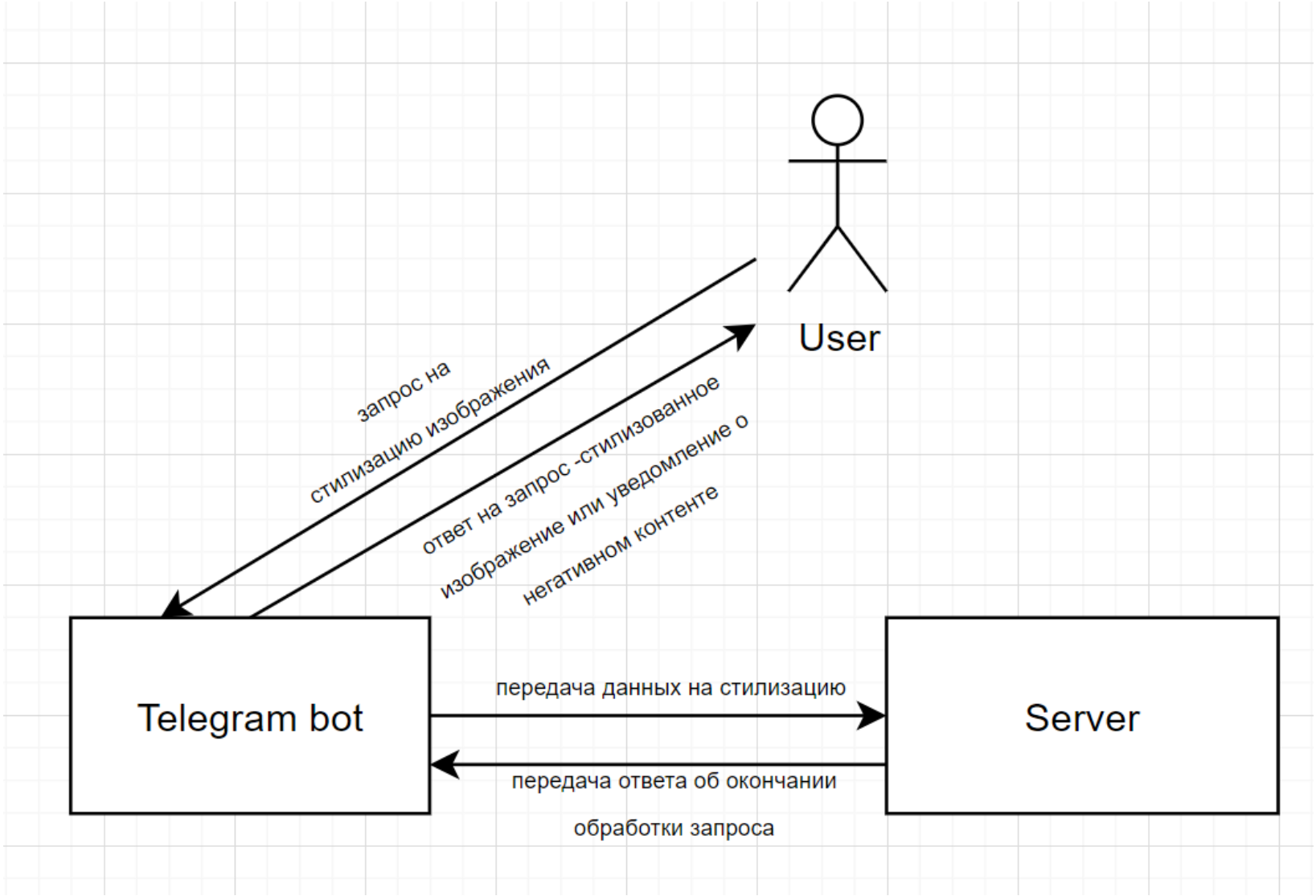
3. Отсутствие зависимости от пользовательских данных:
Классификатор работает как фильтр, который проверяет изображения на неподобающий контент. Поступающие изображения не добавляются в обучающую выборку.

Таким образом, для нашего проекта внедрение дополнительных сервисов для переобучения не требуется. Текущие модели выполняют свои задачи без необходимости в регулярных обновлениях. Если в будущем появится потребность в расширении функциональности (например, добавление новых стилей или улучшение классификации), такие изменения можно реализовать через обучение новых моделей, что не требует постоянной поддержки переобучения.

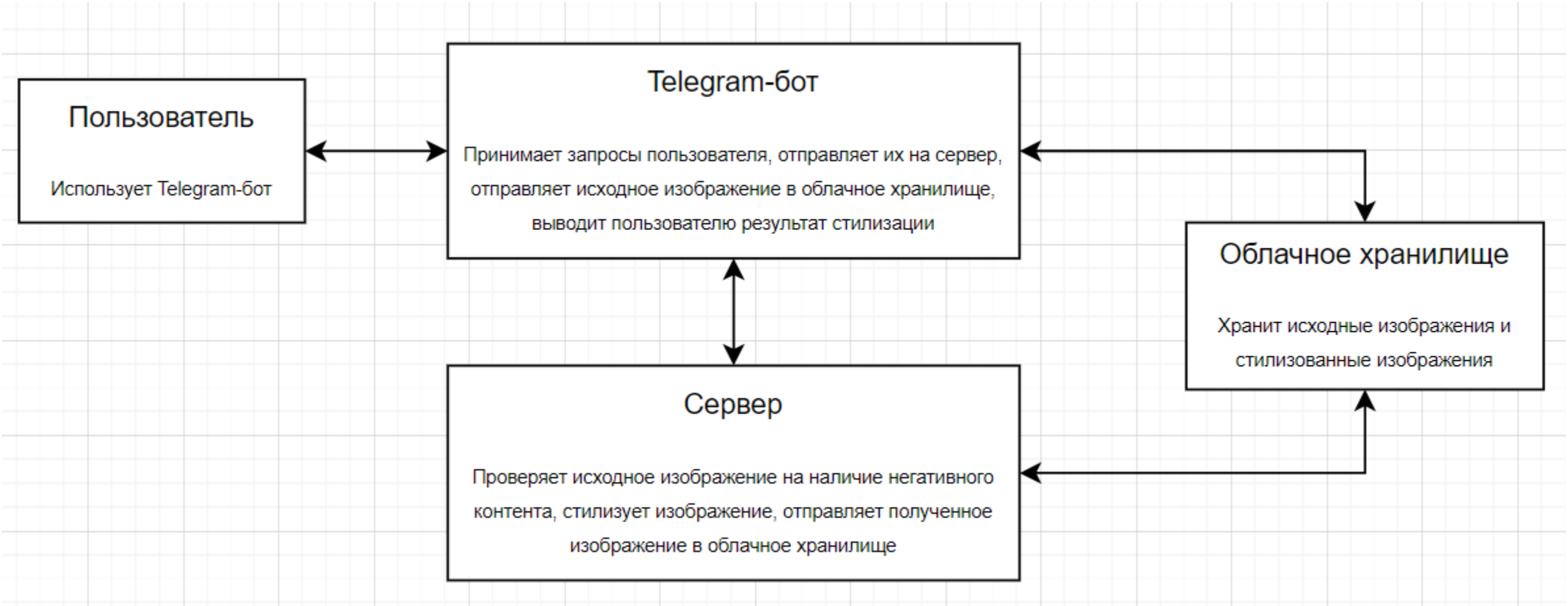
Стек технологий

Компонент	**Технология**	**Обоснование**
Язык программирования	Python	Большое количество библиотек для ML, простота написания кода
ML-фреймворк	TensorFlow, PyTorch	Высокая производительность, поддержка сложных нейронных сетей
Оркестрация	Docker	Для контейнеризации и масштабирования приложения
Оркестрация сообщений	RabbitMQ	Для асинхронной обработки задач, управления очередями сообщений и улучшения производительности взаимодействия между компонентами системы
Хранилище данных	Redis	Для хранения временных данных и быстрого доступа к часто используемой информации
Виртуальная машина	Cloud.ru (Evolution)	Для размещения и масштабирования приложения в облачной среде с высокими показателями доступности и производительности
Объектное хранилище	S3 Evolution Object Storage	Для хранения и управления большими объемами данных с возможностью быстрого доступа и масштабирования

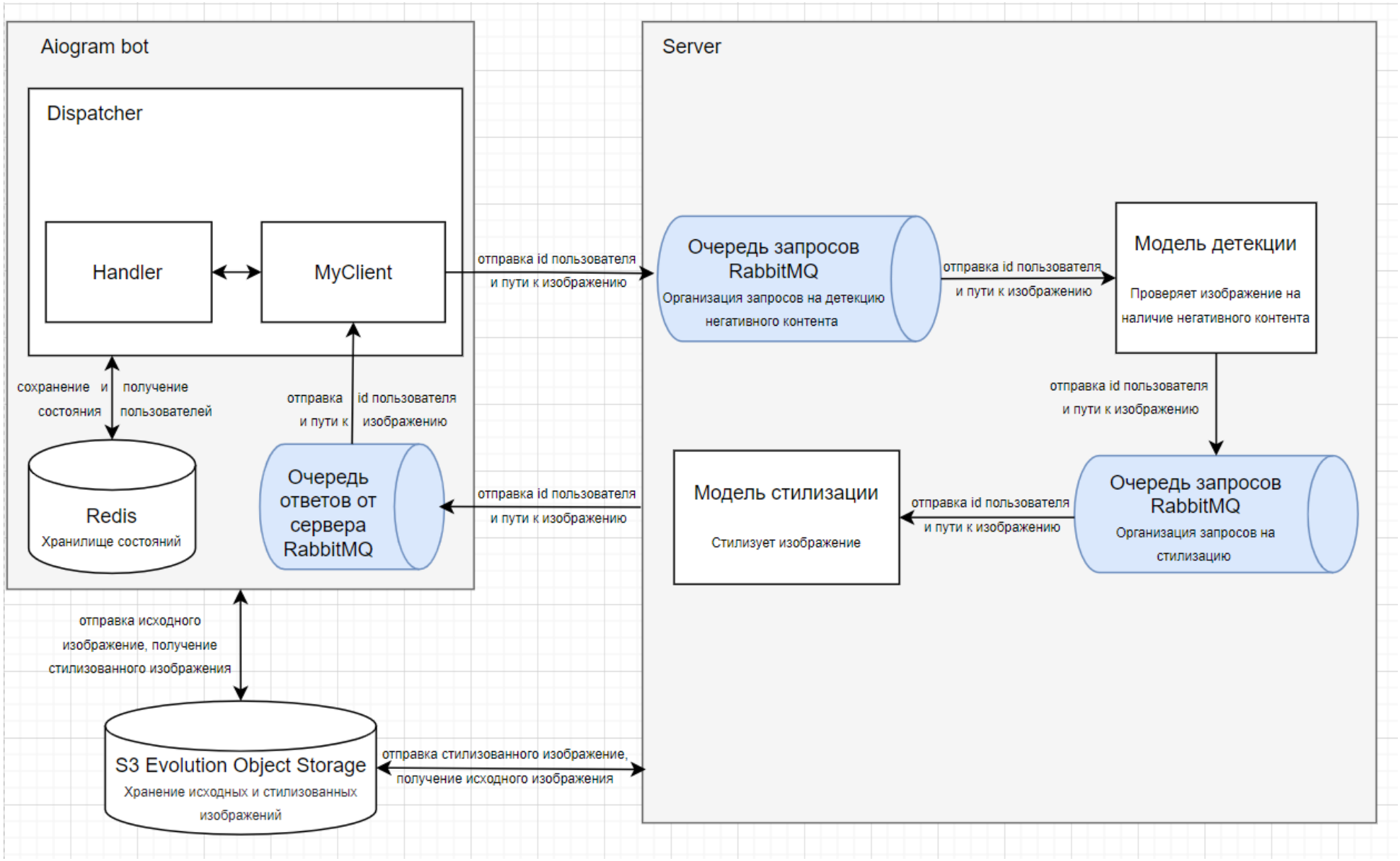
Архитектура (C4 схема). Контекст



Архитектура (C4 схема). Контейнеры



Архитектура (C4 схема). Компоненты



Архитектура (C4 схема). Код

```
/root
│
├── bot/                          # Каталог с кодом Telegram-бота
│   ├── app.py                    # Основной файл для запуска бота
│   ├── client.py                 # Логика взаимодействия с сервером
│   ├── handler.py                # Обработчики команд и событий
│   ├── logging_config.py         # Настройка логирования
│   ├── requirements.txt          # Зависимости для бота
│   └── Dockerfile                # Dockerfile для сборки контейнера бота
│
├── server/                       # Каталог с серверной частью
│   ├── detector.py               # Загрузки модели для детекции негативного контента
│   ├── models.py                 # Загрузка модели для стилизации изображений
│   ├── server.py                 # Основной серверный файл
│   ├── utils.py                  # Вспомогательные функции
│   ├── logging_config.py         # Настройка логирования
│   ├── requirements.txt          # Зависимости для сервера
│   └── Dockerfile                # Dockerfile для сборки контейнера сервера
│
├── docker-compose.yml            # Оркестрация контейнеров
```

Обоснование архитектуры

Архитектура проекта основана на модульности и асинхронности, что позволяет обеспечить высокую производительность, масштабируемость и надежность системы. Выбор технологий был обусловлен как функциональными требованиями проекта, так и преимуществами конкретных инструментов. Ниже приводится обоснование архитектурных решений и выбора технологий.

Язык программирования: Python

Python был выбран за его простоту, читабельность и обширную экосистему библиотек для машинного обучения и обработки данных, таких как TensorFlow, PyTorch, NumPy и PIL. Эти особенности сделали Python оптимальным выбором для быстрой разработки и интеграции ML-решений.

Почему не другой язык? Альтернативы, такие как Java или C++, требуют большего времени на реализацию и имеют меньшую популярность среди библиотек для обработки изображений и нейронных сетей.

ML-фреймворк: TensorFlow и PyTorch

Эти фреймворки предоставляют все необходимое для разработки, обучения и применения сложных нейронных сетей. В проекте VGG16 используется как часть модели для выделения признаков изображений. TensorFlow и PyTorch обеспечивают гибкость в настройке и высокую производительность при работе с большими объемами данных.

Почему не OpenCV или Scikit-learn? Эти библиотеки недостаточно мощны для работы с глубокими нейронными сетями и не поддерживают современные архитектуры, такие как VGG16.

Обоснование архитектуры

Оркестрация контейнеров: Docker

Docker позволяет упаковать приложение вместе со всеми зависимостями, обеспечивая совместимость и упрощая развёртывание в различных средах. Контейнеризация делает систему более управляемой и масштабируемой.

Почему не VirtualBox или подобные технологии? VirtualBox требует значительно больше ресурсов, тогда как Docker работает с контейнерами, которые легче и быстрее.

Брокер сообщений: RabbitMQ

RabbitMQ обеспечивает асинхронное взаимодействие между компонентами системы. Это особенно важно для задач стилизации изображений, которые требуют времени на обработку, и проверки изображений на неприемлемый контент. Асинхронная обработка позволяет разгрузить основные процессы и улучшить отзывчивость бота.

Почему не Celery или Kafka? Celery использует Redis как брокер, что менее эффективно для сложных очередей сообщений. Kafka лучше подходит для потоковой обработки данных, но RabbitMQ проще в настройке и более универсален.

Обоснование архитектуры

Хранилище данных: Redis

Redis используется для временного хранения данных в формате «ключ-значение», таких как информация о запросах пользователей. Этот инструмент отличается низкой задержкой и высокой скоростью обработки запросов.

Почему не PostgreSQL? База данных SQL или NoSQL избыточна для временного хранения информации и не обеспечивает такой же скорости доступа.

Облачная инфраструктура: Cloud.ru (Evolution)

Cloud.ru предоставляет удобную и производительную облачную инфраструктуру, которая позволяет масштабировать проект в зависимости от нагрузки. Высокая доступность сервиса обеспечивает бесперебойную работу бота.

Почему не Google Cloud или подобные инфраструктуры? Cloud.ru предлагает более доступные тарифы и соответствует требованиям локального рынка.

Объектное хранилище: S3 Evolution Object Storage

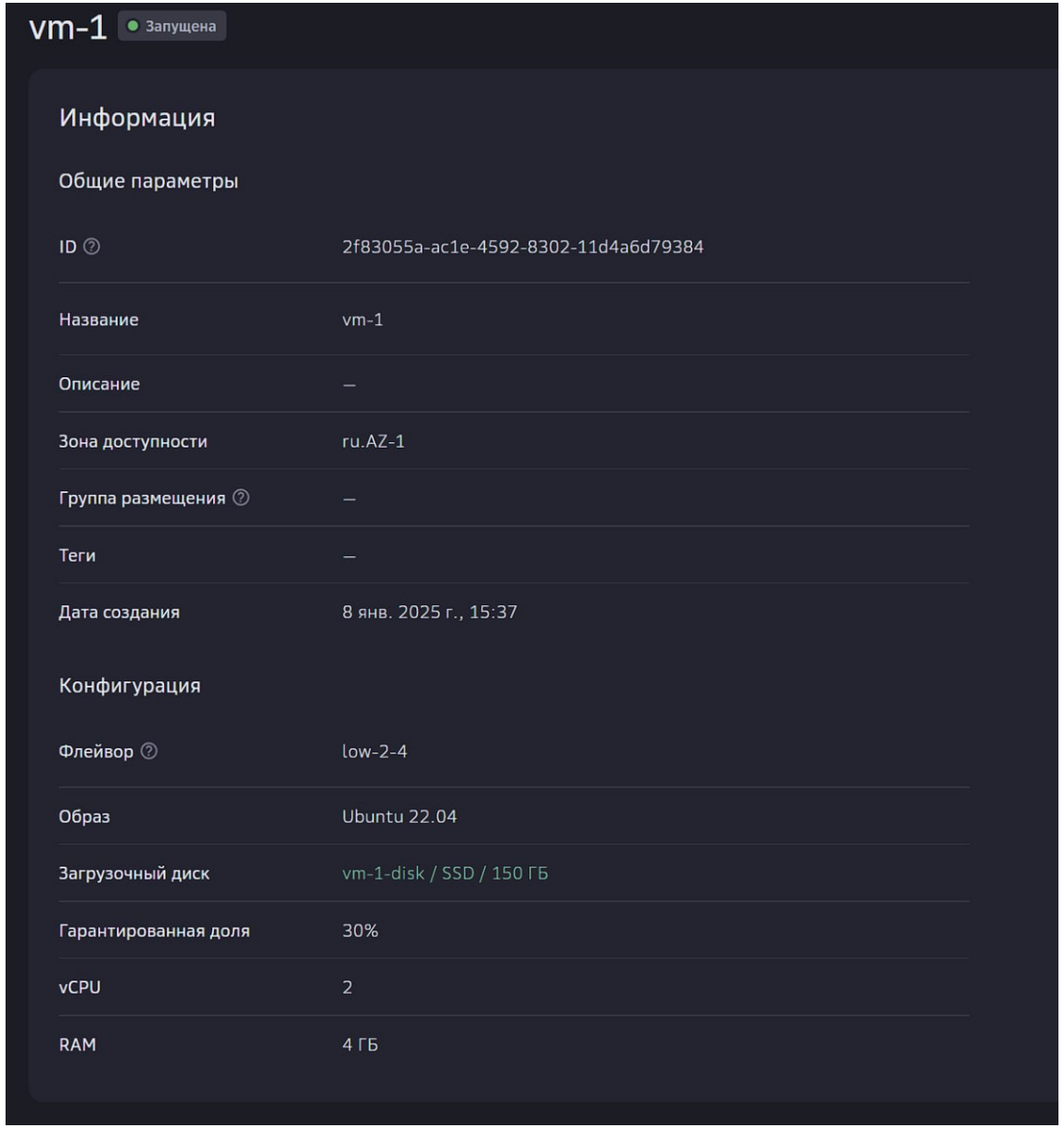
Объектное хранилище используется для долговременного хранения изображений и данных. S3 Evolution поддерживает управление большими объемами данных, предоставляя быстрый доступ и возможность масштабирования.

Почему не локальное хранилище? Локальное хранилище сложно масштабировать и обеспечить высокую доступность данных.

Виртуальная машина

Виртуальная машина на cloud.ru была выбрана для этого проекта, так как она:

- Предоставляет оптимальную производительность для работы моделей и Telegram-бота.
- Обеспечивает централизованное управление и доступность.
- Позволяет эффективно использовать ресурсы и адаптироваться к растущей нагрузке.
- Обеспечивает высокий уровень безопасности, отказоустойчивости и интеграции с другими сервисами.

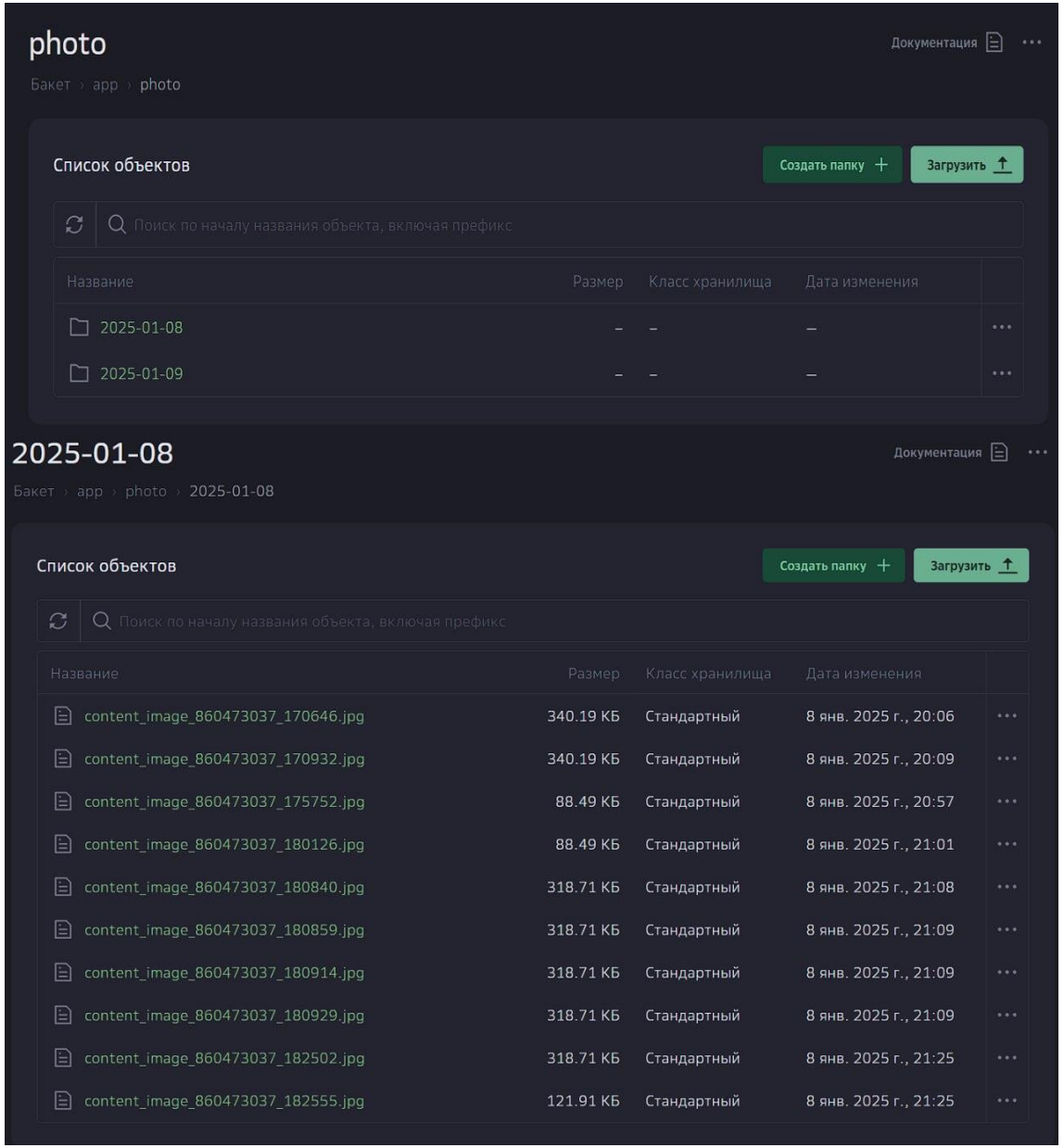


Объектное хранилище

S3-хранилище на cloud.ru используется в проекте для:

- Хранения входных изображений и результатов работы модели.
- Масштабируемого и отказоустойчивого хранения данных.
- Разгрузки локальных ресурсов виртуальной машины.
- Удобной интеграции с компонентами проекта и обеспечения безопасности данных.

Это решение даёт гибкость и надёжность, необходимые для эффективной работы проекта с большим количеством пользователей.



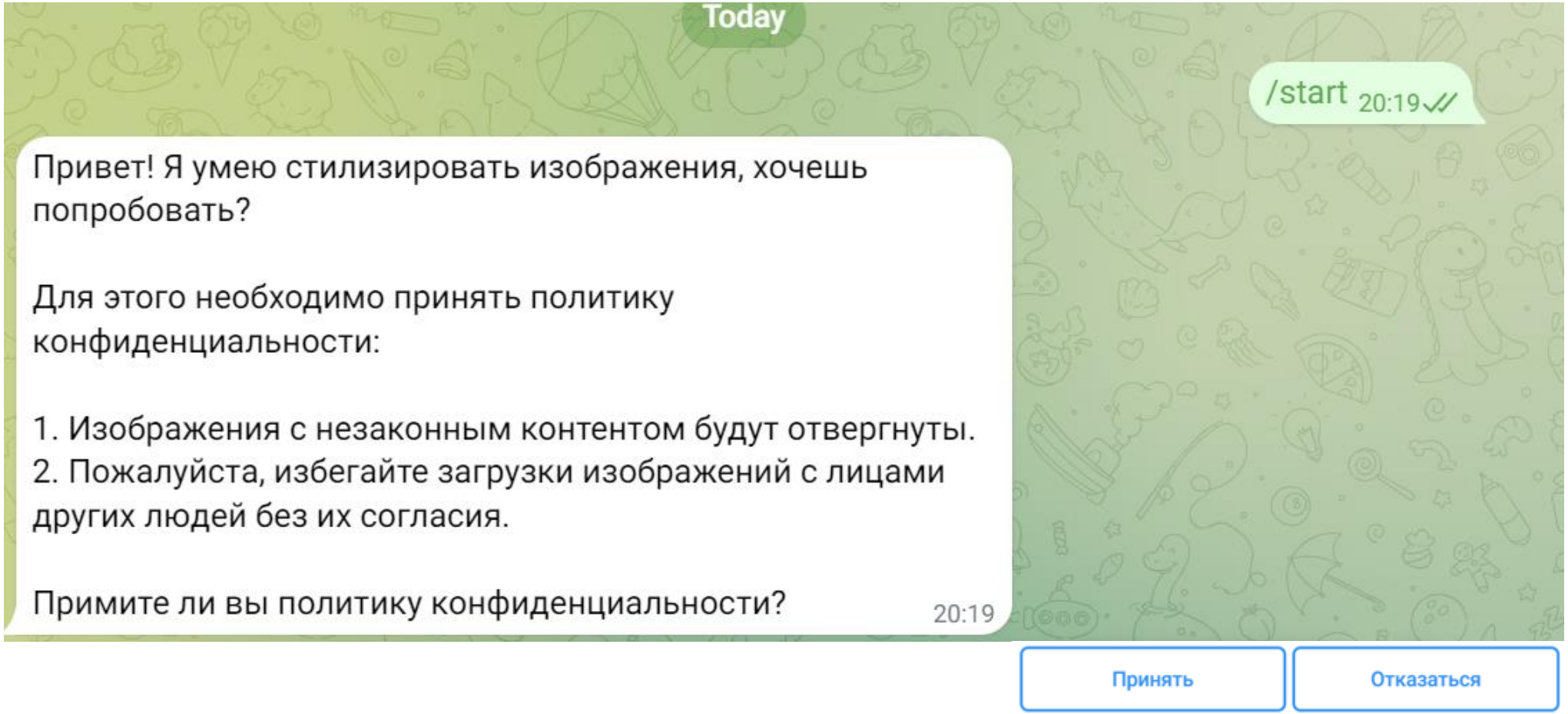
Логирование

В нашем проекте существуют логи бота и логи сервера, все они хранятся локально на сервере в папках bot.log, server.log. В них хранятся логи за текущий и предыдущий дни, а очищаются они раз в сутки в полночь. Ниже приведем ряд причин, почему мы храним логи локально, а не в облачном хранилище s3:

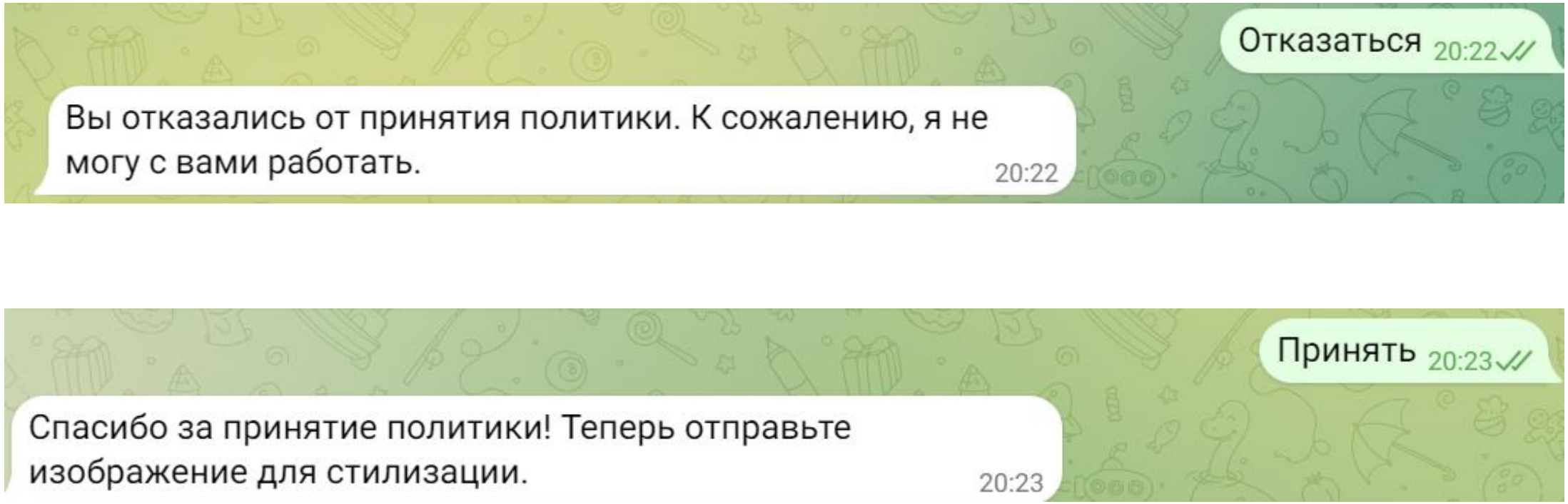
- Наши логи сервера и бота не имеют критической ценности для длительного хранения, поэтому локальное хранение является достаточным решением.
- Локальные логи проще использовать для немедленного анализа (например, через grep, tail, или системные инструменты). При работе с облачными хранилищами может потребоваться дополнительный процесс загрузки логов или настройка инструментов для анализа.

Пример работы бота

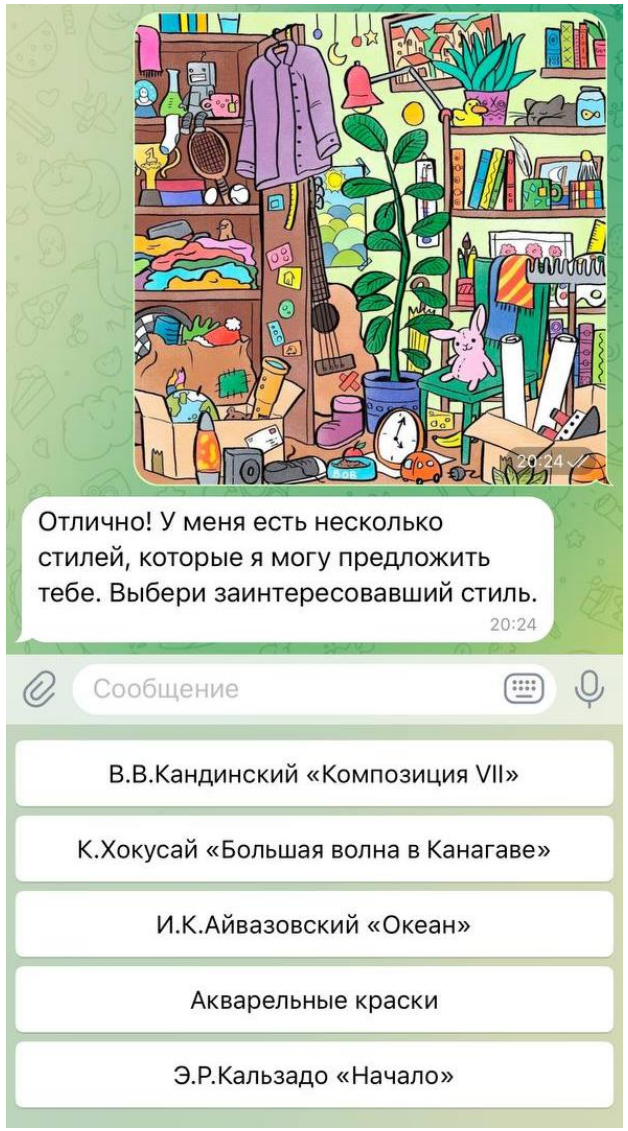
Покажем как функционирует бот по стилизации изображений



Пример работы бота



Пример работы бота



Полезные ссылки

- Модель стилизации
- Модель детекции
- GitHub репозиторий проекта
- Бот StyleTransfer
- Обратная связь (Журавлев Кирилл telegram)

Спасибо за внимание!