

Федеральное государственное автономное образовательное учреждение высшего
образования

Национальный исследовательский университет

«Высшая школа экономики»

Факультет компьютерных наук

Баранова Дарья Дмитриевна

**РАЗРАБОТКА СЕРВИСА ДЛЯ ГЕНЕРАЦИИ РЕАЛИСТИЧНЫХ
ИЗОБРАЖЕНИЙ НА ОСНОВЕ ЭСКИЗА С ПОМОЩЬЮ НЕЙРОСЕТЕВЫХ
МОДЕЛЕЙ**

Выпускная квалификационная работа

студента образовательной программы магистратуры «Машинное обучение и
высоконагруженные системы» по направлению подготовки 01.04.02 Прикладная
математика и информатика

Рецензент
ученая степень, ученое
звание, должность

Руководитель
ученая степень, ученое
звание, должность

Нарек Алвандян
И.О. Фамилия

Москва, 2023 год

Аннотация

Работа содержит 3 главы, 43 страницы, 19 рисунков, 2 таблицы, 19 использованных источников.

ГЕНЕРАЦИЯ ИЗОБРАЖЕНИЙ, SKETCH-TO-IMAGE, ДИФФУЗИОННАЯ МОДЕЛЬ, LATENT GUIDANCE PREDICTOR, TELEGRAM BOT

Целью данной работы является создание метода генерации реалистичного изображения по скетчу с возможностью регулировать степень реализма и схожести со скетчем, а также интегрирование метода в полноценный высоконагруженный сервис.

В первой главе рассматриваются различные существующие подходы к решению задачи генерации изображений на основе скетча. Исследуются главные методы генерации изображений и особенности их реализации. Проводится сравнение и затем выбирается метод, который будет использоваться в работе.

Во второй главе описывается структура всей будущей системы, проводится разделение системы на отдельные компоненты, перечисляются основные необходимые запросы, которые система должна поддерживать, а также описывается, как каждая из компонент системы будет работать.

В третьей главе описываются использованный датасет и способ его получения, проводятся экспериментальные исследования для реализации оптимального метода генерации. Также описываются средства разработки, вспомогательные библиотеки и фреймворки, использованный стек технологий. Формулируются требования к всей системе и реализуются все компоненты сервиса. Приводятся результаты тестирования готовой системы.

Оглавление

Введение.....	4
Глава 1 Анализ существующих подходов.....	6
1.1 Существующие подходы генерации изображений по эскизу	6
1.1.1 GAN.....	6
1.1.1.1 Описание подхода GAN.....	6
1.1.1.2 Conditional Adversarial Networks (pix2pix).....	7
1.1.1.3 Contextual GAN	8
1.1.2 Diffusion model.....	10
1.1.2.1 Описание подхода diffusion model.....	10
1.1.2.2 Stable diffusion model.....	13
1.1.2.3 Classifier Guidance for Latent Diffusion Models	13
1.1.2.4 Diffusion-based framework that generates images from Sketches and Strokes (DiSS). 15	
1.1.2.5 MLP Latent Guidance Predictor	17
1.2 Вывод.....	18
Глава 2 Разработка методов и проектирование системы.....	20
2.1 Проектирование системы в рамках создания Telegram бота.....	20
2.2 Описание способов взаимодействия с Telegram ботом	22
2.3 Вывод.....	24
Глава 3 Программная реализация системы генерации и проведение экспериментов	25
3.1 Используемая модель диффузии.....	25
3.2 Системные требования	25
3.3 Обучающая выборка.....	25
3.4 Построение baseline решения.....	26
3.5 Проведение экспериментов	30
3.5.1 Изменение стартовой точки	30
3.5.2 Отказ от MLP.....	32
3.6 Программная реализация разработанной системы.....	33
3.6.1 Спецификация требований к системе	33
3.6.2 Используемый стек технологий	34
3.6.3 Архитектура системы.....	35
3.7 Тестирование системы.....	37
3.7.1 Скорость работы	37
3.7.2 Обработка некорректных запросов	39
3.8 Вывод.....	40
Заключение	41
Список использованных источников	42

Введение

В настоящее время задача генерации изображений находится на острие развития науки. Множество задач сохраняют свою актуальность и требуют более тщательных исследований несмотря на недавно произошедшие прорывы в области и большое количество существующих нестандартных решений.

Данная работа посвящена задаче Sketch-to-Image – преобразованию заданного эскиза в изображение. Другими словами, необходимо сгенерировать реалистичное изображение на основе наброска (эскиза), сохраняя при этом определенную семантику и границы отдельных элементов. Эта задача играет важную роль в задачах синтеза и обработки изображений, таких как редактирование фотографий и раскрашивание. Синтез изображений на основе эскизов позволяет людям, не являющимся художниками, создавать реалистичные изображения без значительных художественных навыков или специальных знаний в области синтеза изображений. Разработанные решения могут использоваться в качестве мощных инструментов в индустрии художественного дизайна, или, к примеру, в качестве инструмента, помогающего в составлении реалистичного фоторобота подозреваемого.

Сложность задачи заключается в том, что элементы эскиза почти всегда будут сильно отличаться от границ соответствующих реалистичных зарисованных элементов. Поэтому необходимо контролировать генерацию, сохраняя баланс между реализмом и соответствием результата исходному эскизу. Для этого необходимо реализовать подход, который позволит регулировать при генерации уровень реализма и степень соответствия зарисовке. Дополнительной сложностью выступает необходимость генерировать детали, которые отсутствуют на эскизе, но необходимы для реалистичного изображения.

Разработанное решение должно быть прикладным, то есть являться комплексным законченным инструментом – продуктом, который можно использовать.

Таким образом, в качестве объекта исследования выступает готовое изображение, которое было получено в процессе генерации. А предметом исследования выступает процесс получения этого изображения, то есть процесс его генерации с помощью нейросетевой модели на основе заданного наброска.

Целью работы является создание системы генерации изображений на основе эскизов (или зарисовок), сохраняя при этом заданную семантику и задавая определенный уровень реализма.

Поставленная цель предполагает решение следующих задач:

- выполнить обзор существующих методов генерации изображений на основе эскизов;
- получить размеченные обучающие данные;
- разработать метод генерации изображения;
- экспериментально определить оптимальную архитектуру сети и оптимальную реализацию метода генерации;
- реализовать систему генерации в виде полноценного сервиса;
- выполнить тестирование системы.

Глава 1 Анализ существующих подходов

1.1 Существующие подходы генерации изображений по эскизу

Если рассматривать задачу в более общем виде, то можно её свести к Image-to-Image задаче, которая является ключевой во всей области компьютерного зрения. Большинство подходов к решению этой задачи основывается на нескольких базовых архитектурах, таких как GAN и Diffusion models, которые будут описываться далее. Задача Sketch-to-Image не является исключением, поэтому её существующие решения также основываются на указанных моделях с некоторыми модификациями в архитектуре. Рассмотрим подробнее некоторые из них.

1.1.1 GAN

1.1.1.1 Описание подхода GAN

Generative Adversarial Networks (GANs) представляют собой подход к генерации данных с помощью нейронных сетей. При таком подходе используются две нейронные сети – генератор и дискриминатор. Первая сеть – генератор – генерирует необходимые данные, стараясь создать их максимально похожими на обучающие данные. Вторая сеть – дискриминатор – представляет из себя классификатор, который пытается различить между собой обучающие данные и данные, полученные из генератора.

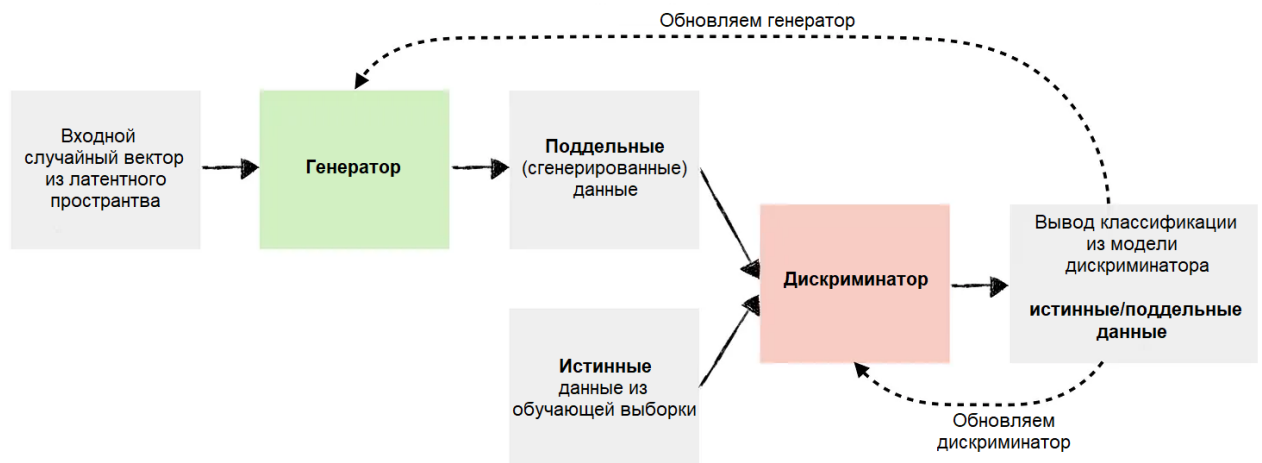


Рис.1 Стандартная архитектура GAN [1].

Весь подход состоит в том, чтобы провести обучение этой пары состязающихся между собой моделей таким образом, чтобы генератор пытался обмануть дискриминатор, создавая все более и более реалистичные данные, а дискриминатор сохранял возможность находить и отличать их от подлинных. Таким

образом, после обучения получим две готовые сети, которые можно использовать в различных задачах. Для задачи генерации данных (в частности, изображений) найдет применение обученный генератор, дискриминатор же будет не нужен. Однако он может быть полезен в других задачах компьютерного зрения, рассмотрение которых выходит за рамки данной работы.

1.1.1.2 Conditional Adversarial Networks (pix2pix)

Стандартную архитектуру GAN можно модифицировать, делая генерацию более управляемым процессом. В частности, для того, чтобы генерировать изображения определенного класса можно на вход моделям дополнительно подавать лэйбл или метку этого класса.

Такая модификация GAN называется Conditional Generative Adversarial Network (cGAN).

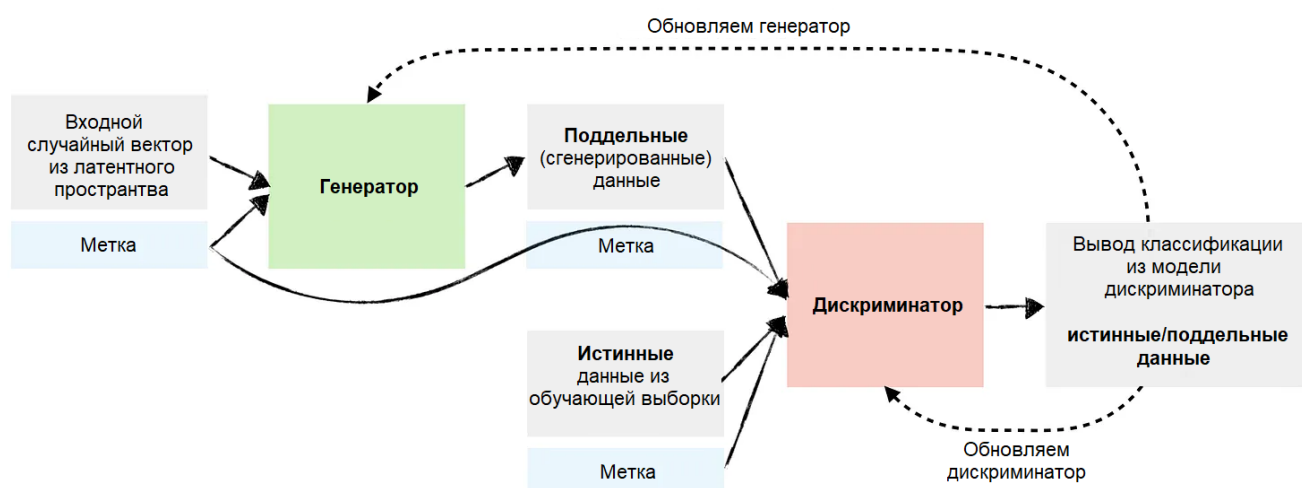


Рис.2 Архитектура условного (conditional) GAN [1].

Входной меткой может выступать что угодно, в частности, это может быть текстовое описание или исходное изображение, которое затем нужно модифицировать. Для задачи генерации реалистичного изображения по наброску входной меткой как раз будет выступать этот набросок.

В работе [2] исследователями использовалась такая архитектура cGAN. При этом, входным изображением не всегда выступал именно набросок, так как задача в исследовании ставилась иначе – создать «переводчик» изображений из одной области в другую. В качестве области может выступать RGB-изображение, карта

границ, карта семантических меток и т. д. Из-за этого входным изображением выступало соответствующее подмножество изображений, содержащих объект в соответствующей области.

Такой подход может частично решать поставленную в работе задачу. Для этого исходной областью будет выступать карта границ объекта, а целевой областью – RGB-изображение (см. рисунок 3).



Рис. 3 Результат работы pix2pix модели при переводе изображения из карты границ в RGB-изображение

Недостатком данного решения является сильное ограничение на соответствие границ объекта на сгенерированном изображении граничным линиям эскиза. Такое ограничение сильно портит качество генерации в задаче генерации по эскизу, так как границы эскиза зачастую условны и редко совпадают с очертаниями соответствующего реалистичного объекта.

1.1.1.3 Contextual GAN

Указанный недостаток был также описан и исправлен в рамках другой более новой работы [3]. По словам авторов, её главное преимущество заключается именно в отсутствии упомянутого строгого ограничения на пиксельное соответствие для границ.

Новый подход заключается в том, что теперь генератор будет пытаться восстанавливать изображение, как схожим образом это происходит в задаче Inpainting. Другими словами, авторы взглянули на задачу с новой стороны – необходимо не сгенерировать изображение, а восстановить его, представив эскиз в качестве поврежденного исходного изображения. В традиционной Inpainting задаче поврежденная часть входного изображения дополняется с использованием

окружающего содержимого изображения в качестве контекста. Однако в нашей задаче контекстом будет выступать эскиз.

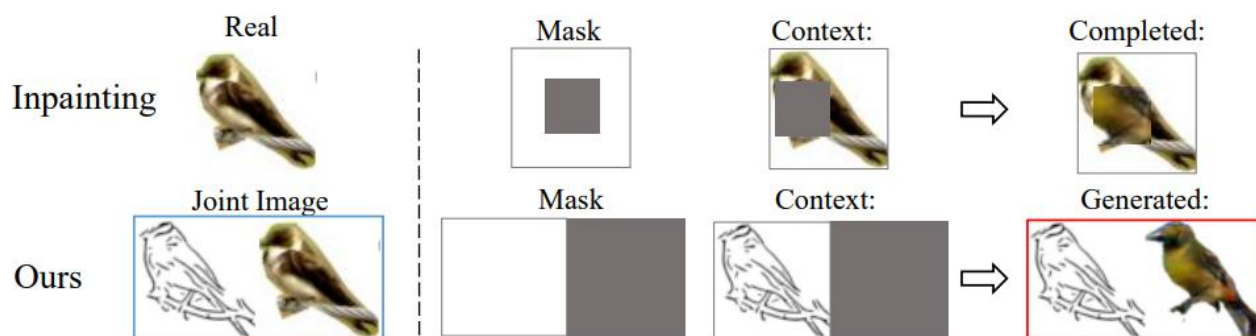


Рис. 4 Иллюстрация подхода Contextual GAN. Эскиз используется в качестве контекста

Для иллюстрации приведен рисунок 4. При дорисовке поврежденной части изображения птицы в верхнем ряду немаскированная часть птицы являются контекстуальной информацией. По аналогии, теперь «испорченной» частью будет выступать все изображение, которое необходимо сгенерировать, а «контекст» будет предоставляется входным эскизом (нижний ряд рисунка 4). При этом входным изображением теперь будет выступать объединенная пара – скетч слева и поврежденная часть изображения справа.

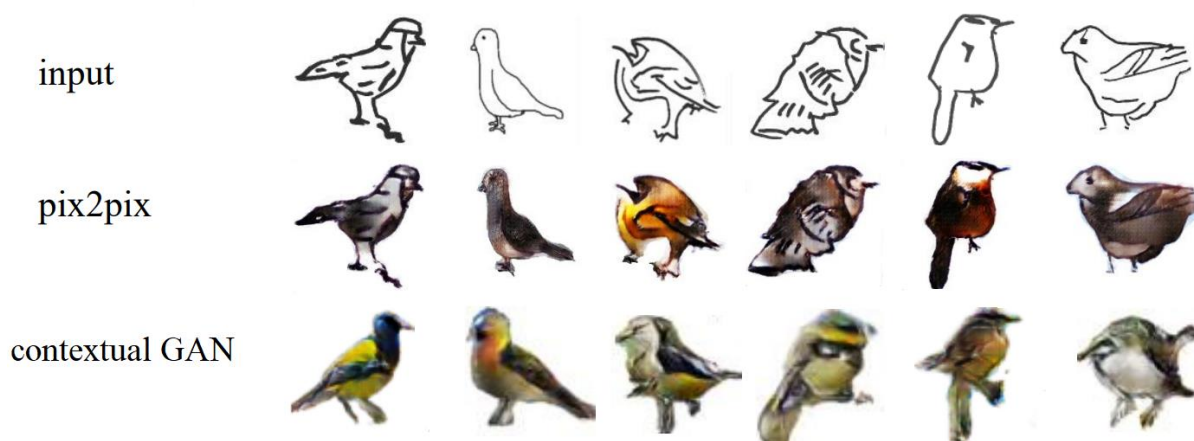


Рис.5 Сравнение результатов работы Contextual GAN и Pix2Pix на примере генерации птиц

Сравнивая полученные результаты с результатами работы pix2pix модели, которая была описана ранее, авторы статьи приходят к выводу, что pix2pix модель имеет тенденцию строго следовать очертаниям входного эскиза, в то время как

Contextual GAN сохраняет устойчивость к искаженным наброскам, так как, по мнению авторов, больше заточена под воспроизведение изученного распределения данных.

1.1.2 Diffusion model

1.1.2.1 Описание подхода diffusion model

Основная идея диффузионной модели состоит в том, что с помощью итеративного прямого диффузионного процесса поданное на вход изображение «разрушается», смешиваясь с шумом до тех пор, пока оно целиком не станет состоять из белого шума. После этого происходит обратный диффузионный процесс – процесс восстановления изображения или его «расшумление», который также происходит итеративно [4]. Таким образом, имея правильно обученную сеть, мы можем подавать ей на вход изображение, состоящее полностью из белого шума, и получать на выходе сгенерированное изображение. Рассмотрим подробнее прямой и обратный диффузионные процессы.

Прямой процесс заключается в том, что исходное изображение x_0 постепенно зашумляется, проходя определенное количество итераций $t=\{0,...,T\}$, пока на последней итерации T изображение x_T не начнет содержать только белый шум, то есть пока не будет полностью состоять из данных нормального распределения $N(0, I)$. Таким образом, обозначим как $q(x_t|x_{t-1})$ функцию зашумления изображения x_{t-1} до состояния x_t .

При обратном процессе изображение, наоборот, расшумляется, проходя итерации от x_T до x_0 . На шаге i модель получает изображение x_i на вход, и возвращает его в менее зашумленном состоянии x_{i-1} , таким образом постепенно генерируя его. Итеративность процесса генерации позволяет добиться лучшего качества, а также отслеживать и контролировать генерацию на каждом шаге. Обозначим $p(x_{t-1}|x_t)$ как функцию расшумления изображения, то есть перевода из состояния x_t в состояние x_{t-1} .

Согласно введенным обозначениям, будем иметь:

$$q(x_t|x_{t-1}) = N(x_t, \sqrt{1 - \beta_t}x_{t-1}, \beta_t I),$$

где:

$N(\cdot)$ – нормальное распределение,

$\sqrt{1 - \beta_t}x_{t-1}$ – матожидание,

$\beta_t I$ – дисперсия,

β_t – коэффициент, отвечающий за постепенное изменение шума в изображении, $\beta_t \in [0, 1]$.

Также можно свернуть все итеративные переходы, соединив все преобразования в одну итерацию и избавившись от промежуточных вычислений:

$$q(x_t|x_0) = N(x_t, \sqrt{\bar{a}_t}x_0, (1 - \bar{a}_t)I),$$

где:

$$\alpha_t = 1 - \beta_t,$$

$$\bar{a}_t = \prod_{s=1}^t \alpha_s.$$

Теперь подробнее опишем обратный процесс.

$$p(x_{t-1}|x_t) = N(x_{t-1}, \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)),$$

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{a}_t}} \epsilon(x_t, t) \right),$$

где:

$$\epsilon(x_t, t) - \text{шум, } \epsilon \sim N(0, I)$$

$\Sigma_\theta(x_t, t)$ считается зафиксированной величиной и может быть установлена как β_t (в дальнейших работах этот параметр будет также учиться моделью).

Для реализации описанного процесса необходимо создать сеть, которая будет предсказывать шум, который присутствует на изображении [5]. После этого на каждой итерации необходимо вычитать предсказанный шум из входного изображения, делая его таким образом менее зашумленным. Функция потерь L_t будет являться стандартной среднеквадратичной ошибкой:

$$L_t = \|\epsilon - \epsilon(x_t, t)\|^2$$

Можно выразить x_{t-1} через x_t :

$$x_{t-1} = \begin{cases} \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{a}_t}} \epsilon(x_t, t) \right) + \sqrt{\beta_t} \epsilon, & t > 1 \\ \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{a}_t}} \epsilon(x_t, t) \right), & t = 1 \end{cases}$$

Algorithm 1 Training	Algorithm 2 Sampling
1: repeat 2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ 3: $t \sim \text{Uniform}(\{1, \dots, T\})$ 4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 5: Take gradient descent step on $\nabla_{\theta} \ \epsilon - \epsilon_{\theta}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, t)\ ^2$ 6: until converged	1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 2: for $t = T, \dots, 1$ do 3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$ 4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_{\theta}(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$ 5: end for 6: return \mathbf{x}_0

Рис.6 Алгоритмы для процедуры обучения и выборки для диффузионной модели

Нужно заметить, что количество шума на каждой итерации может и будет отличаться, так как это позволяет достичь лучшего качества. При линейном характере изменения доли шума на изображении наблюдается несколько проблем, а именно:

1. На последних итерациях процесса, когда изображение приближается к тому, чтобы стать белым шумом и утратить последние элементы содержательной информации, визуальные изменения уже не кажутся значительными (см. рисунок 7). Можно сказать, что последняя треть или даже половина итераций уже выглядит как полностью зашумленное изображение.

2. При этом, на первых итерациях зашумления ключевая содержательная информация слишком быстро утрачивается.



Рис.7 Зашумление при линейном (вверху) и косинусном (внизу) подходе соответственно на каждой итерации t от 0 до T . В последней четверти линейного графика изображения представляют собой почти чистый шум, тогда как косинусный график добавляет шум медленнее.

Исходя из этого, в качестве альтернативы был предложен [6] косинусный характер изменения доли шума на изображении.

Архитектура сети представляет собой U-Net. На вход сети также будет подаваться номер итерации, представленный особым образом. Это важная деталь, так как, как было указано выше, в зависимости от номера итерации на изображении будет присутствовать разная доля шума.

1.1.2.2 Stable diffusion model

Stable Diffusion model является усовершенствованием классической diffusion модели. Она является latent diffusion model (LDM), что означает, что модель работает не с изображениями напрямую, а с их закодированными представлениями в латентном пространстве.

Stable Diffusion состоит из 3 частей: вариационного автоэнкодера (VAE), U-Net и дополнительного текстового энкодера, так как генерирует изображение на основе не только изображения, но и текстового описания.

Кодировщик VAE сжимает изображение из пространства пикселей в скрытое пространство меньшего размера, сохраняя основную семантику изображения и отбрасывая наименее фундаментальную информацию. Это позволяет увеличить скорость работы сети, так как размер данных уменьшился. Декодировщик VAE генерирует выходное изображение, преобразуя представление обратно в пространство пикселей.

Для эмбединга текста используется предварительно обученная сеть CLIP [7].

1.1.2.3 Classifier Guidance for Latent Diffusion Models

Одним из существующих подходов, помогающим делать генерацию данных с помощью диффузионных моделей более управляемой, является использование вспомогательного «руководящего» классификатора в процессе генерации (так называемый метод Classifier Guidance). Подход состоит в том, что при генерации помимо диффузионной модели также используется дополнительная модель-классификатор, loss которой может показывать, насколько генерируемое изображение действительно соответствует входным данным (например, текстовому описанию или эскизу, заданным на входе). При таком подходе производная полученной ошибки классификатора будет использоваться для обновления изображения после каждого шага генерации.

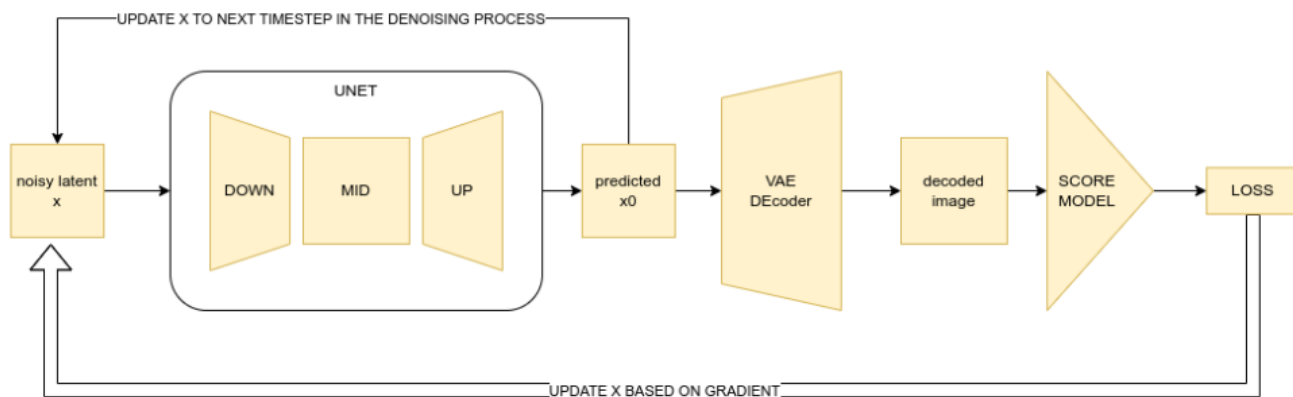


Рис.8 Стандартная реализация Classifier Guidance для латентных диффузионных моделей [8]

Для реализации такого обучения требуются большие вычислительные мощности и большое время работы, так как предполагается, что обратных проброс градиентов будет являться сложной и дорогостоящей операцией (требуется вычисление градиентов в модели классификации, затем декодере VAE (в случае моделей скрытой диффузии) и самой диффузионной U-Net на каждом этапе генерации).

Один из способов обойти это ограничение был описан в работе [9]. Исследователи используют в качестве альтернативы две диффузионные модели – conditional и unconditional, и используют их вместе в процессе генерации. В работе утверждается, что такой подход дает схожие результаты, что и Classifier Guidance, однако требует сильно меньше вычислений и ресурсов. В качестве гиперпараметра будет выступать вес (пропорция) смешивания описанных моделей. Утверждается, что задав гиперпараметр оптимально можно достигнуть того же качества, что и с помощью дополнительной модели классификатора.

Также, две диффузионные модели можно совместить в одну, задав безусловную диффузию как модель, обусловленную нулевым токеном \emptyset . Именно такой подход и был использован в работе экспериментально. Обучение такой скомбинированной модели тогда будет происходить следующим образом. Гиперпараметром будет задаваться p_{uncond} – вероятность подать на вход нулевой токен вместо определенного условия c для conditional модели. Случайным образом с вероятностью p_{uncond} будет задаваться вход на каждой итерации обучения.

При генерации шум будет вычисляться как комбинация выходов conditional и unconditional моделей:

$$\tilde{\epsilon}_{\theta}(\mathbf{z}_{\lambda}, \mathbf{c}) = (1 + w)\epsilon_{\theta}(\mathbf{z}_{\lambda}, \mathbf{c}) - w\epsilon_{\theta}(\mathbf{z}_{\lambda})$$

1.1.2.4 Diffusion-based framework that generates images from Sketches and Strokes (DiSS) [10]

В данной работе поставленная задача генерации фотореалистичных изображений на основе эскиза решается с помощью описанной выше диффузионной модели. В работе описан подход, при котором на вход подается эскиз, а также цветовая карта, которая является изображением такого же формата, что и изображение эскиза, но состоит из трех каналов и содержит в себе цветовую информацию, то есть своеобразную карту, где какой цвет должен располагаться на итоговом изображении. Пример приведен на рисунке ниже.



Рис.9 Пример входных и выходных данных в работе DiSS.

При этом стандартная диффузионная модель имеет несколько модификаций:

1. На вход модели (U-Net) подаются одновременно представление изображения эскиза в латентном пространстве и представление цветовой карты.
2. За основу метода обучения взят описанный выше подход, использующий комбинацию из conditional и unconditional моделей. Он был немного изменен под особенности конкретной задачи. Обучение состояло из двух этапов. Первый – обучение при полных данных, когда на вход подавались как признаки эскиза в латентном пространстве c_{sketch} , так и признаки цветовой карты c_{stroke} . На втором этапе какое-то из условий убиралось (заменялось на пустой токен \emptyset), задавая

своеобразную unconditional модель. Тогда комбинация выходов приобретала следующий вид:

$$\begin{aligned}\hat{\epsilon}_{\theta}(x_t, t, \mathbf{c}_{\text{sketch}}, \mathbf{c}_{\text{stroke}}) &= \hat{\epsilon}_{\theta}(x_t, t, \emptyset, \emptyset) \\ &+ s_{\text{sketch}}(\hat{\epsilon}_{\theta}(x_t, t, \mathbf{c}_{\text{sketch}}, \emptyset) - \hat{\epsilon}_{\theta}(x_t, t, \emptyset, \emptyset)) \\ &+ s_{\text{stroke}}(\hat{\epsilon}_{\theta}(x_t, t, \emptyset, \mathbf{c}_{\text{stroke}}) - \hat{\epsilon}_{\theta}(x_t, t, \emptyset, \emptyset)).\end{aligned}$$

С помощью этой линейной комбинации контролируется степень соответствия генерируемого изображения каждому из условий (условию близости к эскизу и условию близости к карте цветов)

3. Степень реализма достигается путем использования метода итеративного уточнения скрытых переменных (iterative latent variable refinement), описанного в работе [11]. Метод заключается в том, чтобы на каждом шаге генерации изображения x_{t-1} из x_t вычислять итоговое изображение x'_{t-1} как линейную комбинацию x_{t-1} , $\mathcal{L}_N(x_{t-1})$ и $\mathcal{L}_N(c_{\text{comb}t-1})$, где $\mathbf{c}_{\text{comb}t-1} \sim q(\mathbf{c}_{\text{comb}t-1} | \mathbf{c}_{\text{comb}0})$ с $\mathbf{c}_{\text{comb}0} = \mathbf{c}_{\text{comb}}$, при этом c_{comb} – это сумма изображения эскиза и цветовой карты, а \mathcal{L}_N представляет собой линейную операцию фильтрации низких частот (linear low pass filtering operation), которая заключается в том, чтобы понизить размерность изображения до размерности N и затем увеличить размерность изображения до исходного. Коэффициент N будет закладывать степень схожести итогового изображения с входными данными, определяя «насколько много частот входного изображения» будет учитываться при генерации итогового.

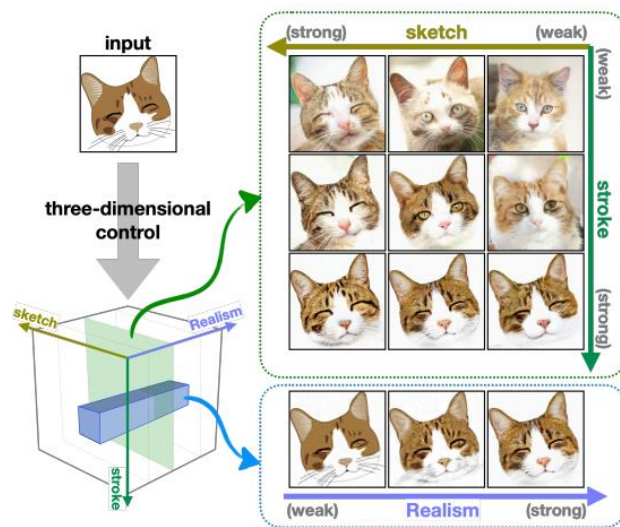


Рис.10 Пример полученных результатов работы модели DiSS при разных соотношениях соответствия эскиз-цвет-реализм

1.1.2.5 MLP Latent Guidance Predictor [12]

В данной работе использовалась stable diffusion model, в которой также присутствовала модификация авторов. Основная идея подхода в том, что к готовой диффузионной модели вида text-to-image добавляется вспомогательная MLP сеть, которая будет выступать помощником в процессе генерации изображения.

Метод позволяет сгенерировать изображение на основе эскиза и текстового описания. В результате могут получиться разные результаты для одного эскиза – разница будет формироваться на основе различного текстового описания.



Рис.11 Примеры различной генерации при одном эскизе и разном текстовом описании

Можно сказать, что MLP в данном случае будет выступать в роли Classifier Guidance. Его задача заключается в учете границ эскиза, который предварительно переведен в латентное пространство, поэтому для его реализации не потребуется раскодировать изображение из пространства признаков. Добавленный MLP обучается предсказывать истинный эскиз по переданным ему активационным слоям диффузионной модели. На выходе требуется, чтобы перцептрон выдавал представление максимально похожее на представление эскиза. Формально, для входного тензора изображения w и контекста c на итерации t обозначим соединенные (сконкатенированные) активации выбранных внутренних слоев сети $\{l_1, \dots, l_n\}$ как:

$$F(w|c, t) = [l_1(w|c, t), \dots, l_n(w|c, t)]$$

Тогда входное измерение MLP представляет собой сумму количества каналов выбранных слоев. На выходе ожидается, что перцептрон вернет изображение аналогичного размера, которое получается на выходе диффузионной сети после очередной итерации.

Также перцептрон должен уметь работать с входными данными с любой итерации диффузионной модели, поэтому дополнительно ему на вход подается номер итерации t .

После того, как перцептрон будет обучен, сеть будет считаться обученной полностью и готовой для генерации. Для того, чтобы на итерации t , получив входное изображение x_{t-1} , сгенерировать изображение x_t , необходимо, чтобы диффузионная сеть предсказала шум $\epsilon(x_t, t)$, который присутствует на изображении. Во время этого предсказания у модели должны считаться и сохраняться активации $F(w|c, t)$ выбранных внутренних слоев, которые используются MLP. На их основе перцептрон генерирует соответствующие латентное представление эскиза E^\sim , вычисляется мера схожести полученного эскиза E^\sim на истинный e : $L(E^\sim, E(e)) = \|E^\sim - E(e)\|^2$, а затем считается соответствующий антиградиент $-\nabla_{x_t} L$. Тогда новое изображение на выходе будет рассчитываться как:

$$\widetilde{x_{t-1}} - 1 = x_{t-1} - \alpha * \nabla_{x_t} L$$

Где:

α – коэффициент, который задает степень схожести итогового изображения на эскиз. Эмпирически было выяснено, что лучше себя показывает нормированный коэффициент по формуле:

$$\alpha = \frac{\|x_t - x_{t-1}\|_2}{\|\nabla_{x_t} L\|_2} \beta$$

В этом случае коэффициентом выступает уже β , который будет иметь порядок $O(1)$.

Таким образом, диффузионная модель будет стараться соблюдать указанные границы эскиза при генерации. Исходной моделью до модификации выступает предобученная latent diffusion model (Stable Diffusion).

1.2 Вывод

В этой главе было проведено исследование нескольких существующих подходов для решения задачи генерации реалистичного изображения на основе скетча. На основе сравнения результатов можно сделать ряд выводов. Во-первых, можно заметить, что качество изображений, полученных диффузионными

моделями, сильно превышает качество изображений, сгенерированных моделями GAN архитектуры. Поэтому будущее решение будет основываться на архитектуре диффузионной модели. Во-вторых, решение 1.1.2.5 представляется более предпочтительным по сравнению с решением 1.1.2.4, так как кажется более простым и интересным подходом. При этом, это самое новое исследование, у него до сих пор не опубликован исходный код и недоступны веса обученной модели. В работе описаны хорошие полученные результаты, а также решение удовлетворяет сформулированным требованиям, а именно – существует механизм контроля степени соответствия эскиз-реализм. Обучение модели занимает достаточно мало времени, так как используется предобученная модель диффузии, и необходимо обучить только вспомогательную MLP сеть.

Таким образом, решение предлагается строить на основе архитектуры, описанной в исследовании [12].

Глава 2 Разработка методов и проектирование системы

В данной главе будет описана подробнее предлагаемая архитектура будущей системы. Основой системы будет выступать Telegram бот, получающий и обрабатывающий запросы пользователей. Для выполнения генерации необходимо сделать запрос к боту, передав необходимые данные. После того, как генерация будет завершена, бот отправит пользователю результат генерации новым сообщением.

2.1 Проектирование системы в рамках создания Telegram бота

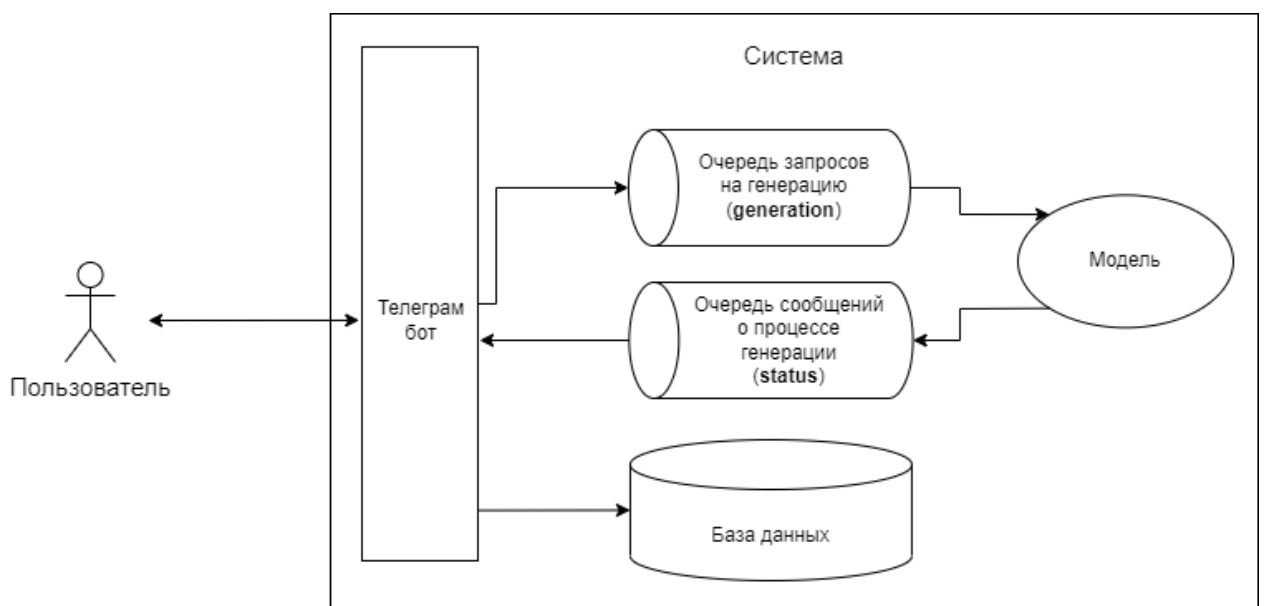


Рис.12 Функциональная схема системы

Система будет состоять из нескольких больших компонентов, взаимодействующих между собой. Общая схема системы изображена на рисунке выше.

Работа системы начинается с момента первого взаимодействия пользователя с ней. Система поддерживает разные типы взаимодействий, но для описания основного функционала будем предполагать, что в данном случае речь идет о запросе на генерацию изображения, так как в этом случае обработка будет осуществляться каждым компонентом системы.

Запрос от пользователя на генерацию изображения должен сопровождаться двумя обязательными параметрами – загруженным изображением скетча и

текстовым описанием контента на изображении. Если оба параметра переданы, то запрос будет принят системой в обработку. Все взаимодействия с системой пользователь осуществляет через интерфейс Telegram бота. Доступ до любых других компонентов системы напрямую для пользователя невозможен.

Приняв запрос, система перенаправляет его из бота в нейросетевую модель для генерации. Это происходит через очередь сообщений “generation”. Общение между ботом и моделью происходит через две очереди – “generation” и “status”. Такой тип общения был выбран по нескольким причинам:

1. Это позволяет улучшить масштабируемость системы. В случае, если необходимо увеличить скорость генерации моделью можно легко добавить несколько аналогичных экземпляров нейросетевой модели в систему. Тогда количество параллельно работающих обработчиков запросов увеличится и появится возможность справляться с большей нагрузкой на систему. При этом нет необходимости даже реализовывать какой-либо балансировщик нагрузки, чтобы определять, какое сообщение на какой экземпляр модели попадет – брокер очереди сообщений по умолчанию будет выполнять балансировку в соответствии с настроенной стратегией.

2. Увеличивается надежность системы. В случае, если модель выйдет из строя или будет недоступна связь с ней, то запрос на генерацию не пропадет, а останется лежать в очереди сообщений до тех пор, пока связь с моделью не будет восстановлена и запрос не будет успешно обработан.

3. Очередь позволяет максимально разделить независимые логически компоненты системы и осуществлять асинхронное общение, избегая таймаутов при ожидании ответа.

В системе присутствуют две очереди – для отправки сообщений от бота к модели (“generation”) и от модели к боту (“status”). Очереди имеют названия, описывающие основную цель отправляемых сообщений:

- “Generation” – отправка сообщений для начала генерации изображения моделью. Сообщение будет включать в себя необходимые для генерации данные, как например текстовое описание и название входного файла с эскизом.

– “Status” – отправка сообщений с информацией о статусе происходящей генерации. Сообщение будет включать в себя информацию о том, на каком этапе генерации находится модель, сколько было сгенерировано и сколько осталось. Информация о том, что генерация завершена, а также данные, хранящие в себе имя полученного файла, будут передаваться от модели боту также через эту очередь.

Последним неописанным компонентом системы является база данных. С ней осуществляет взаимодействие бот на всех этапах обработки запроса. В базе данных хранится информация о пользователе, от которого поступали запросы, о самих запросах, и о статусе их выполнения.

После того, как генерация завершена и сообщение об это получено ботом через очередь “status”, бот отправит пользователю сообщение, а также прикрепит к нему сгенерированное изображение.

2.2 Описание способов взаимодействия с Telegram ботом

Опишем виды запросов от пользователя, которые умеет обрабатывать бот. Если запрос не входит в указанный список, то он будет проигнорирован.

1. Запрос на генерацию изображения. Сообщение должно выглядеть как отправленное изображение эскиза с текстовым описанием, которое передается как подпись к загружаемому изображению. Если запрос сформирован правильно, то после его принятия системой будет сгенерировано изображение, которое вернется клиенту в качестве нового ответного сообщения. Максимальное количество принятых и обрабатываемых запросов не может превосходить 3 запросов для одного пользователя.

2. Тестовый запрос на /start. Создан для того, чтобы иметь возможность проверить работоспособность бота без создания запроса на генерацию. Если бот запущен и исправен, то на запрос должен вернуться ответ в виде приветствия пользователя.

3. Запрос с описанием инструкций для работы с ботом /help. Создан для предоставления вспомогательной информации о том, какие функции выполняет бот, какие запросы принимает, и как запросы должны быть сформированы.

4. Запрос на изменение параметра генерации init_k. Запрос является вспомогательным, позволяющим настраивать параметры генерации изображения

моделью. Параметры настраиваются и сохраняются для конкретного пользователя. Если параметры не заданы, то будут использованы дефолтные значения. Данный запрос позволяет настраивать параметр `init_k`, который контролирует формирование изначального шумного изображения, поступающего на вход диффузионной модели. Более подробно данный параметр описан в главе 3.5.1, где указан как параметр `D`.

5. Запрос на изменение параметра генерации `grad_k`. Аналогично предыдущему запросу, является вспомогательным, позволяющим менять параметры генерации. Коэффициент регулирует степень влияния вычисленного градиента на генерируемое изображение. Представляет собой параметр `beta`.

Для каждого из описанных выше запросов возвращается соответствующее сообщение бота с ответом на запрос. Опишем подробнее некоторые ответные сообщения:

1. Сообщение об ошибке. В процессе обработки запроса может возникнуть несколько предусмотренных ошибок, для каждой возвращается соответствующий текст сообщения. Например, при превышении максимального количества поступивших и не успевших обработаться запросов на генерацию изображения будет возвращаться сообщение с текстом «You have done too many requests. Please wait until some generation will be finished».

2. Сообщение о том, что прикрепленное изображение сохранено и запрос на генерацию принят. После того, как бот получил запрос на генерацию, и загрузил прикрепленное изображение, будет вернут соответствующий ответ, информирующий пользователя об этом.

3. Сообщение о том, что генерация началась. После того, как будет отправлено предыдущее сообщение, в базе данных сохранена необходимая информация о запросе, а также в очередь сообщений “generation” помещено соответствующее сообщение о начале генерации, будет отправлено сообщение о том, что генерация началась.

4. Сообщение о статусе генерации. Данное сообщение будет отправлено несколько раз по мере выполнения генерации. Точнее, под этим сообщением можно понимать обновление текста ранее отправленного сообщения. После того, как генерация изображения была начата моделью, в качестве реакции на каждое новое

поступающее боту сообщение из очереди “status” будет отправлено сообщение со статусом генерации – на сколько итераций генерации было совершено в процентном соотношении.

5. Сообщение о том, что генерация завершена. Будет отправлено в тот момент, когда бот получит сообщение из “status” очереди о том, что генерация выполнена на 100% и будет загружено итоговое сгенерированное изображение с сервера. Данное сообщение представляет собой отправку готового изображения пользователю с подписью, что генерация завершена.

6. Сообщение о том, что параметры генерации успешно обновлены. Отправляется на каждый успешный запрос на обновление параметров /init_k или /grad_k. Если в процессе обработки запроса произошла какая-либо ошибка, то вместо этого сообщения будет отправлено сообщение с ошибкой (пункт 1).

2.3 Вывод

Система для генерации будет строиться на основе Telegram бота. Она будет состоять из нескольких компонентов и иметь асинхронное общение между ботом и нейросетевой моделью, что позволит достичь легкой масштабируемости всей системы в случае больших нагрузок. Общение будет осуществляться через две очереди. Бот должен уметь обрабатывать несколько перечисленных различных запросов, а также выполнять базовую проверку запросов на корректность их формата и данных.

Глава 3 Программная реализация системы генерации и проведение экспериментов

В данной главе будет описан подробнее процесс реализации описанного метода для генерации изображения по переданному скетчу, проведенные эксперименты с моделью, а также детали реализации всей системы в целом.

3.1 Используемая модель диффузии

В качестве основной диффузионной модели, на которой будет строиться генерация изображений, была выбрана Stable Diffusion v1-5 [13]. Это latent diffusion модель, в которой используется предварительно обученный кодировщик текста (CLIP ViT-L/14).

3.2 Системные требования

Для локальной работы модель требует достаточно много видеопамяти. В процессе локального запуска на собственном оборудовании (NVIDIA GeForce RTX 3070, 8GB) модели не хватило памяти. Поэтому был арендован специальный сервер с более современными видеокартами, обладающими большим объемом памяти. Можно сказать, что минимальные требования для модели составляют около 12GB видеопамяти для запуска и генерации.

Таким образом, работа с моделью осуществлялась на машине с 1 видеокартой RTX 3090 24 GB VRAM.

3.3 Обучающая выборка

Во всех экспериментах использовался датасет Imagenet [14] с именами классов в качестве текстового описания. Соответствующие изображения эскизов были сгенерированы с помощью модели предсказания краев из [15], а затем дискретизированы с порогом 0,5.

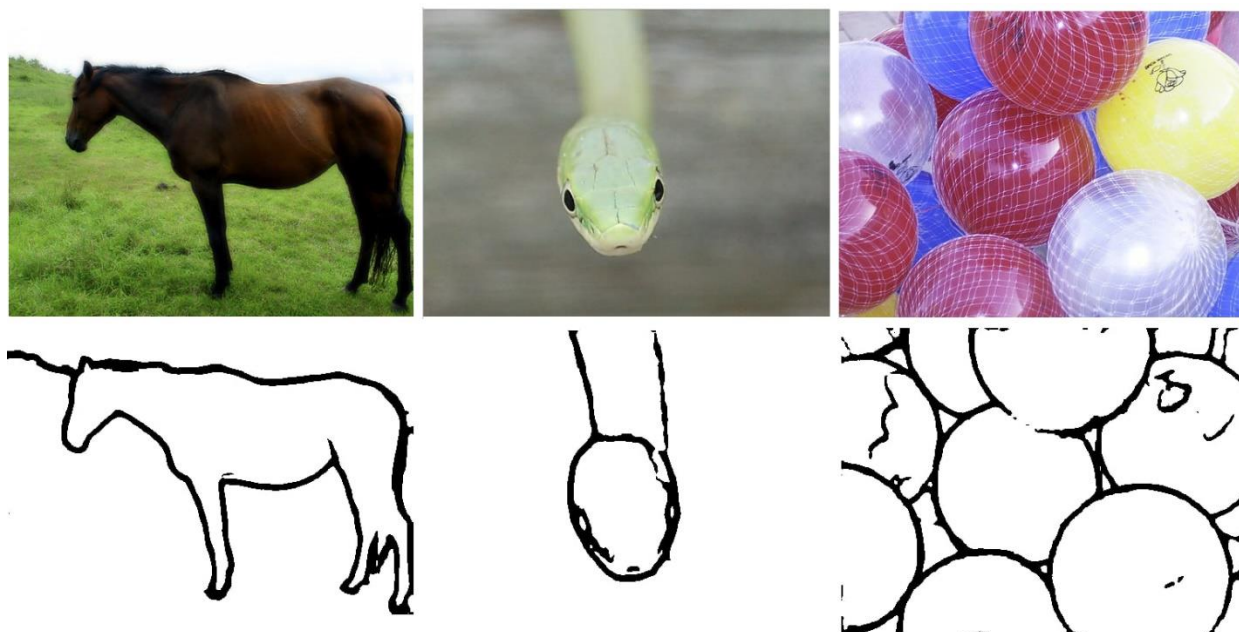


Рис.13 Примеры из обучающей выборки. Сверху оригинальное изображение из датасета Imagenet, внизу – соответствующее обработанное изображение-эскиз

Обучение проходило на подвыборке из 40 тысяч изображений разных классов (300 классов, по 100-200 изображений на каждый класс). В процессе обучения на вход модели подавалось оригинальное изображение, зашумленное до итерации t , а также текстовое описание, которое представляло собой название класса, к которому принадлежит изображение в датасете Imagenet.

3.4 Построение baseline решения

Baseline решением будет считать то, которое описано в статье [12] и в первой главе работы. Решение строилось только на основе опубликованной статьи, так как ни код, ни модель авторы пока не опубликовали.

3.4.1 Построение и обучение MLP

Описанная MLP модель представляет собой guidance модель, которая будет «направлять» (регулировать/контролировать) процесс генерации диффузионной модели. На вход MLP принимает активации с 9 слоев диффузионной модели. Это 3 слоя с input блока модели (2-ой, 4-ый и 8-ой слои), также 3 слоя с middle блока

модели (0-ой, 1-ый и 2-ой слои) и 3 слоя с output блока (также 2-ой, 4-ый и 8-ой слои). Эксперименты с использованием различных слоев, отличных от перечисленных, не принесли значимых улучшений, поэтому в дальнейшем будет описана только перечисленная комбинация.

Так как описанные слои имеют различную размерность, то их необходимо отмасштабировать. Все слои были увеличены до размера референса с помощью билинейной интерполяции и затем объединены вместе вдоль размерности своих каналов. Таким образом, имеем область размера $64*64$, представленную в нескольких каналах, собранных с разных слоев диффузионной модели. Дополнительно на вход MLP подается временная метка, которая хранит в себе номер итерации генерации изображения, а также positional encoding, который представляется как $\sin(2 * \pi * t * 2^{(-l)})$, где $l = 0, \dots, 9$. Таким образом размерность входных данных MLP составляет 9280 канала для $64*64$ изображения. На выходе от модели ожидается тензор, совпадающий по размерности с латентным представлением скетч-изображения (4 канала для $64*64$ изображения).

Обучение MLP проводилось в течение часа на A-100 видеокарте на описанном ранее наборе данных. После работа сети была протестирована на нескольких изображениях (см результаты на рисунке ниже).



Рис.14 Исходное изображение (верхний ряд), соответствующий полученный по нему скетч из обучающего датасета (средний ряд) и скетч, полученный от MLP (нижний ряд)

Можно сказать, что получено достаточно хорошее качество работы MLP. В некоторых случаях предсказанный скетч оказывается даже лучше, чем скетч из обучающей выборки (например, для изображения мандарина по центру – скетч из датасета обрезает верхнюю границу фрукта, в то время как MLP выделяет эту границу, что является более корректным поведением).

3.4.2 Результаты

Для тестирования полученной модели использовались как скетчи из датасета, так и несколько тестовых изображений из датасета SketchyCOCO [16].

Были перебраны различные комбинации параметров реалистичности β и S (доля итераций, на которых будет использоваться вспомогательная guidance модель). В результате чего были обнаружены некоторые описанные ниже закономерности.

Для большого значения β и $S=t$ (вспомогательная guidance модель применяется на всех итерациях генерации) генерируемое изображение получается полностью соответствующим скетчу и теряет всякий реализм. Некоторые примеры представлены на рисунках ниже.



Рис.15 Примеры генерации при разных параметрах β и S

Скетч, на основе которого проходила генерация, представлен на рисунке в первой колонке (черно-белый). Полученные скетчи представлены в остальных колонках. Можно заметить, как при увеличении β становятся более явными линии границ эскиза на итоговом изображении. При этом при уменьшении β границы скетча все менее и менее заметны, но изображение становится более наполненным. Появляется фон и детали, изображение становится более реалистичным.

Однако можно заметить, что несмотря на наличие на изображении четких границ скетча, основной генерируемый контент изображения все равно не совпадает с этими границами. Рассмотрим эту проблему подробнее на примере велосипеда. Соответствующие сэмплы представлены в нижнем ряду рисунка. От диффузионной модели требовалось сгенерировать фотографию красного велосипеда. Можно видеть, что на итоговых генерациях действительно присутствуют красные «пятна», однако они не совпадают с видимыми границами скетча. При значениях β меньше 1.7 заметно, что красный велосипед генерируется в другой части изображения, но при этом его границы как будто размываются. Словно сеть начинает рисовать велосипед независимо от скетча, а потом на уже достаточно больших итерациях, когда основной контент изображения уже заложен, начинает

прорисовываться скетч. Однако его границы уже сильно далеки от основных элементов изображения и модель уже не способна это исправить.

Можно также рассмотреть генерацию при заданном значении $\text{betta} = 1.9$ и различном значении S . Примеры генерации представлены на рисунке ниже.



Рис.16 Примеры генерации при разном параметре S

Видно, что при $S \leq 0.6 * t$ теряется видимость скетча. При этом независимо от значения S также наблюдается несогласованность положения видимого красного велосипеда на изображении с видимыми границами скетча. При $S = 0.5 * t$ скетч уже становится совсем не заметен и не вносит никакого вклада в итоговое изображение. Конечно, это граничное значение может меняться при изменении betta (при увеличении betta скетч перестанет быть заметен уже не при $S=0.5 * t$, а при меньшем значении). Однако приведенные цифры все еще достаточно интересны, так как baseline изначально проектировался с оптимальными параметрами $\text{betta} = 1.6$ и $S = 0.5 * t$, как это было описано в исследовании.

3.5 Проведение экспериментов

3.5.1 Изменение стартовой точки

На основе описанной в предыдущем подпункте проблемы можно выдвинуть гипотезу, объясняющую возникновение этой проблемы. Заметим, что генерация основной сцены изображения происходит на самых начальных итерациях. Чем больше итерация, тем больше прорабатываются детали и меньше закладывается основной контент. Одновременно с этим заметим, что на самых начальных итерациях MLP хуже всего способен выявлять границы будущего скетча, так как большая часть изображения состоит пока преимущественно из шума (или только из него).

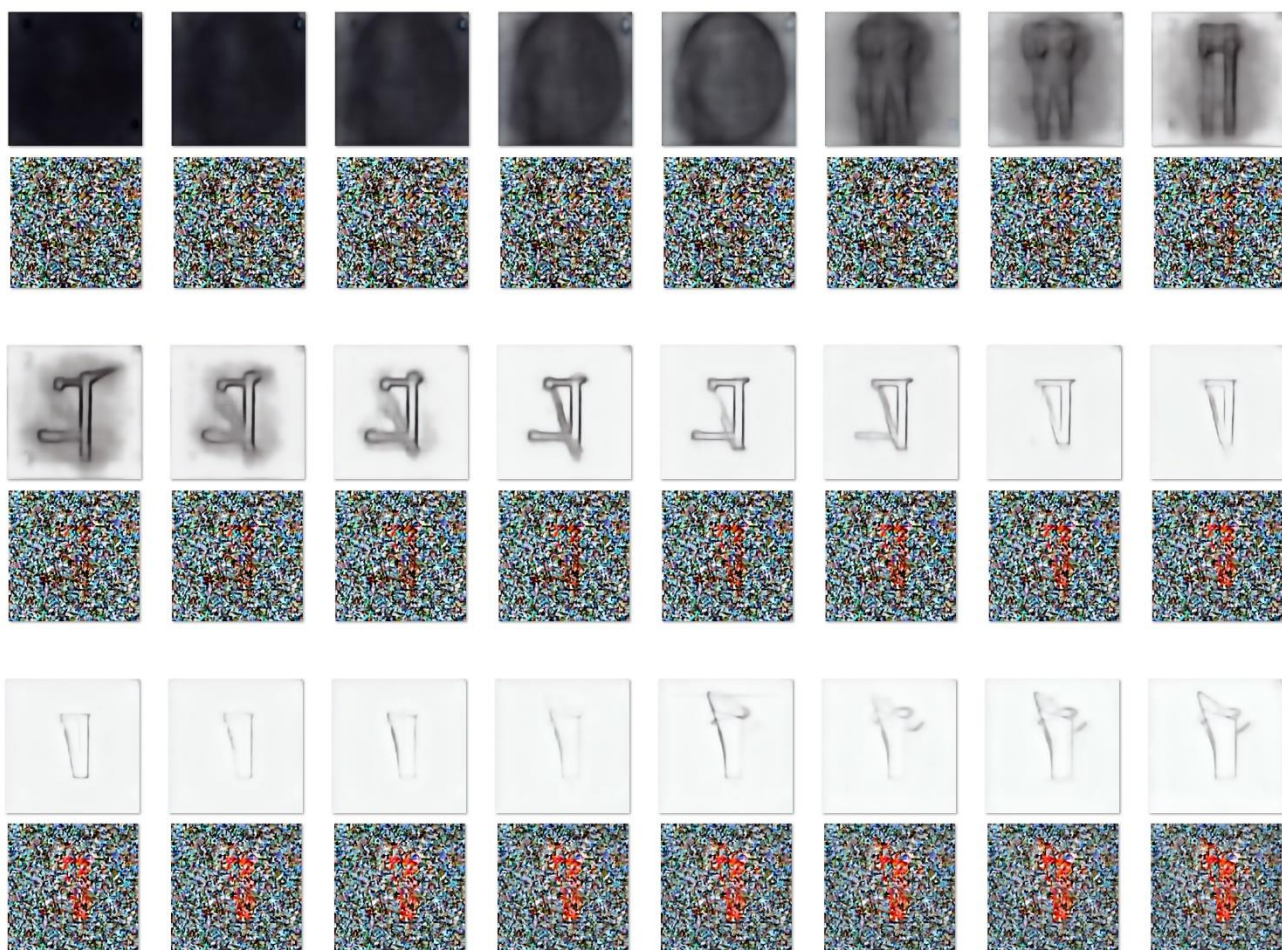


Рис.17 Первые 24 итерации генерации изображения. Слева-направо, сверху-вниз порядок увеличения итераций. Нижний ряд – генерируемое изображение, верхний ряд – полученный с помощью MLP скетч на соответствующей итерации

Таким образом, проблема заключается в том, что MLP хуже всего работает тогда, когда он больше всего нужен. При этом дополнительное дообучение MLP на начальных итерациях не дает никаких значимых результатов.

В качестве решения был использован самый простой метод – добавление на первой итерации к шуму, подающемуся на вход диффузионной модели, предварительно смешанный с шумом входной скетч в некотором соотношении. Результаты представлены на рисунке ниже.

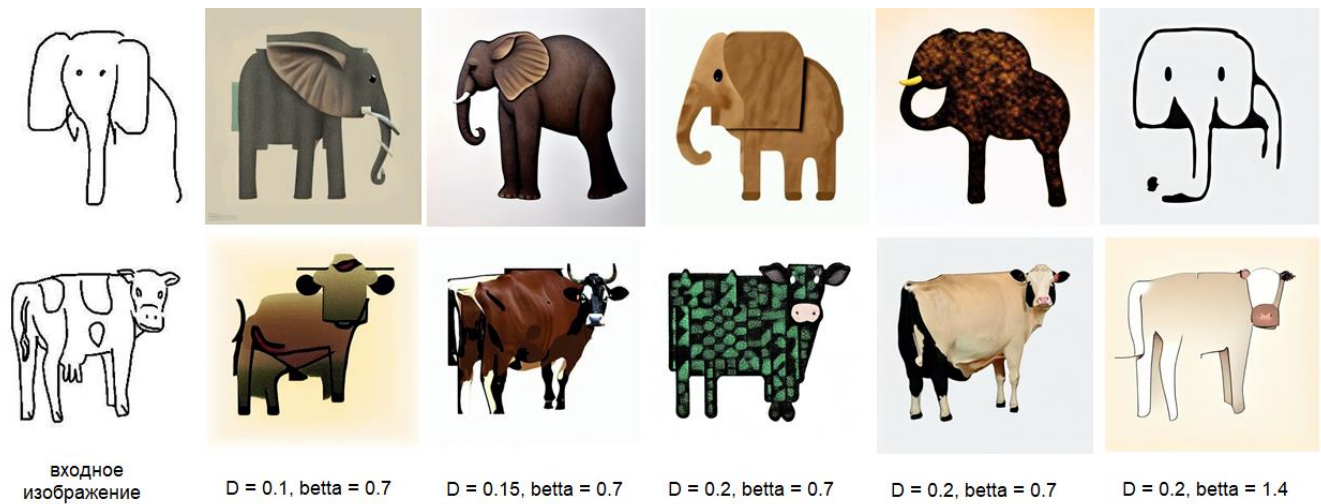


Рис.18 Примеры генерации при разном параметре D и β

Введем дополнительный параметр D , отвечающий за смешивание шума и входного скетча на первой итерации:

$$x_t = (1 - D) * \text{noise} + D * \text{sketch}$$

Тогда получим, что чем больше параметр D , тем сильнее должно быть влияние скетча на итоговое изображение. Примеры, приведенные выше, получены при разных D и β , однако можно заметить, что при увеличении значения D фон изображения становится белым (аналогично фону скетча). Этот недостаток был описан также в исследовании [17] при сравнении работы с моделью SDEdit [18], которая работает схожим образом – в определенных пропорциях смешивает скетч и шум, а затем подает на вход диффузионной модели.

Заметим, что полученные результаты сильно превосходят предыдущие, когда генерация строилась только на MLP. Таким образом, мы улучшили результат baseline модели.

3.5.2 Отказ от MLP

Предположим, что полученное неплохое качество обусловлено только инициализацией, в то время как использование MLP только портит модель. Попробуем исключить перцептрон и использовать для соответствия скетчу только добавление его к шуму на первой итерации. Полученные результаты представлены ниже.

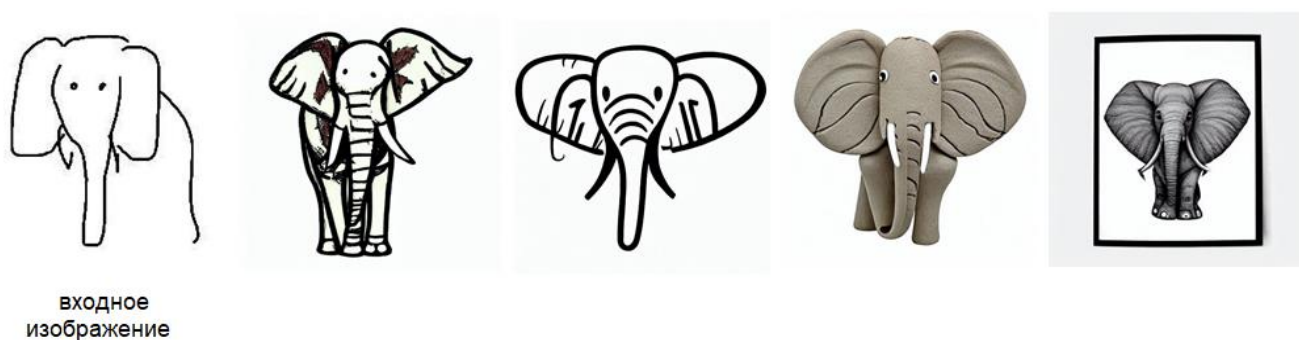


Рис.19 Примеры генерации без использования MLP

Видим, что продолжает преимущественно белый фон. При этом сами границы генерируемых объектов сильно отделились от границ скетча. Больше всего это заметно по ушам слона, которые теперь стали расходиться сильно шире и становиться более круглыми, выходя за границы скетча, где они были более квадратной формы.

Таким образом, использование MLP в качестве guidance predictor позволяло лучше следить за следованием границам, чем при отсутствии перцептрона, когда от границ остается только намек, заложенный на самой первой итерации и деформированный на всех последующих.

3.6 Программная реализация разработанной системы

3.6.1 Спецификация требований к системе

Опишем подробнее требования в системе.

- Для решения задачи необходимо реализовать сервис, который принимает изображение-скетч и строку текста и возвращает в ответ сгенерированное изображение;
- Критическими параметрами системы является правильность её работы и время её работы;
- Сервис должен выдерживать большое количество обращений (быть высоконагруженным);
- Сервис должен быть легко масштабируемым (возможность запустить в нескольких параллельных воркерах);
- Сервис должен иметь возможность менять параметры генерации без остановки работы;

- Сервис должен иметь лимит на количество одновременных запросов от одного пользователя;
- Основные ресурсы при работе сервиса должны тратиться на работу нейросетевой модели. Время работы самого сервиса не должно вносить значительный вклад в общее время обработки запроса;
- Основная функция сервиса – выполнять генерацию изображения по переданному скетчу и текстовому описанию. Сервис может не поддерживать проведение дообучения модели, так как это является слишком сложной операцией, требующей большого объема ресурсов.

3.6.2 Используемый стек технологий

Для решения поставленной задачи было принято решение разрабатывать асинхронный telegram бот. Асинхронность позволит оптимально распределять время для обработки запросов в условиях существования частей системы, которые требуют значительных временных затрат на работу.

Для более понятного описания преимуществ асинхронности обычно приводится пример с игрой в шахматы одновременно с несколькими соперниками. Пусть имеет гроссмейстера, который играет одновременно с 24 соперниками. На ход он должен тратить по 5 секунд, в то время как соперники – по 55 секунд. Рассмотрим два варианта игры – синхронный и асинхронный. В случае синхронной игры гроссмейстер играет одну игру до завершения партии. Пусть игра требует 30 ходов. Она займет $(5 + 55) * 30 = 1800$ секунд или 30 минут. Всё мероприятие займет $24 * 30 = 720$ минут или 12 часов. В случае асинхронной игры гроссмейстер получает возможность перемещаться от одного стола к другому и играть несколько партий параллельно. Пока соперники размышляют над своим ходом, гроссмейстер перемещается к другим столам и делает ходы в других играх. Тогда 1 ход в каждой игре займет $24 * 5 = 120$ секунд или 2 минуты. Всё мероприятие уместится в $120 * 30 = 3600$ секунд или один час.

В качестве языка программирования для реализации telegram бота был выбран Python 3 из-за простоты написания кода и удобства разработки. В качестве

основного фреймворка для бота используется `aiogram`. Это самый популярный асинхронный вариант, рекомендуемый многими разработчиками [19].

В качестве базы данных будет использоваться PostgreSQL с асинхронным клиентом `asynpg`. Фреймворком для более простой и удобной работы с базой данных будет выступать ORM библиотека `SQLAlchemy 2.0`. Брокером сообщений был выбран `RabbitMQ`. На нем будут строиться обе очереди сообщений (“generation” и “status”).

3.6.3 Архитектура системы

Сервис представляет собой бота, который запускается на определенном порте или сокете и входит в бесконечный цикл событий - `event loop`. Сервис имеет свой конфигурационный файл, в котором задаются необходимые параметры, константные значения, ссылки или пути до файлов.

Все приложение состоит из нескольких уровней абстракции:

- `Handlers` – данный уровень содержит описание всех команд, поступающих к боту, которые должны быть им обработаны. Если к боту поступает команда, не описанная в `handlers`, то она будет проигнорирована. Этот уровень абстракции содержит в себе всю бизнес логику бота. При наличии сложных кусков кода, связанных с определенной сущностью базы данных и обладающих функциональной целостностью, они могут быть вынесены в отдельные функции в `Managers` слой. Уровень может содержать вызовы всех других уровней;

- `Managers` – данный уровень должен содержать в себе весь функционал, связанный с работой с определенной сущностью базы данных. Для каждой сущности базы данных существует соответствующий менеджер, который позволяет выполнять связанные с этой сущностью операции. Может содержать вызовы других уровней: `Queries`, `Validators`;

- `Queries` – данный уровень должен содержать в себе только функционал, описывающий запросы к БД. Обязательно подключается для каждого менеджера и содержит в себе базовые функции для запросов к базе. Слой может легко масштабироваться при необходимости. Это происходит путем переопределения базового класса `BaseQuery` новым кастомным классом, содержащим реализацию

какой-то дополнительной функции, выполняющий сложный описанный через ORM запрос. Не может содержать вызовы никаких других уровней;

- **Validators** – данный уровень должен содержать в себе все проверки и валидации. Обязательно подключается для каждого менеджера и содержит в себе стандартный набор проверок, описанный в классе **BaseValidator**. Может быть расширен через переопределение базового класса, аналогично слою **Queries**. Может содержать в себе только вызовы уровня **Queries**.

Схема базы данных состоит из трех таблиц:

1. **User** – содержит в себе информацию о каждом пользователе сервиса, который сделал хотя бы один запрос к нему. Хранит в себе данные о имени и фамилии пользователя, **id** его аккаунта в telegram, а также кастомные параметры генерации, если они хотя бы раз были заданы пользователем.

2. **Photo** – содержит в себе информацию о изображении скетча, которое было передано в запросе пользователя на генерацию изображения. Для каждого нового запроса будет создана новая запись в базе, содержащая информацию о **id** загруженного файла в хранилище telegram, текстовом описании (**prompt**), размере файла и некоторых других его характеристиках.

3. **PhotoToUser** – представляет собой таблицу-связку между таблицами **User** и **Photo**. Дополнительно хранит в себе статус обработки запроса, **id** чата и сообщения, в котором сообщается о статусе запроса, и название итогового файла, когда генерация будет завершена.

Сервис также содержит в себе модуль **queue**, отвечающий за работу с очередями **RabbitMQ**. Модуль состоит из двух классов:

- **Producer**. Помещает данные в очередь генерации “**generation**”. Каждое новое сообщение будет являться триггером для начала новой генерации. Отправляемые данные включают в себя параметры генерации (кастомные параметры пользователя или дефолтные, если пользователь их не задал), имя файла, содержащего скетч, а также текстовое описание.

- **Consumer.** Получает сообщения из очереди “status”. В каждое сообщение помещается процентное значение готовности генерации, а также, если генерация была завершена, то имя сгенерированного файла.

Еще одним заслуживающим внимание модулем является `middlewares`. Он состоит из двух `middleware`:

- **SessionMiddleware.** Прокидывает новую сессию базы данных для каждого нового запроса к боту.

- **ExceptionMiddleware.** Так как в процессе работы бота, а также в процессе валидации поступивших запросов могут возникать ошибки, нужно уметь их корректно обрабатывать. Корректной обработкой будем считать наличие сообщения от бота пользователю, которое оформлено как ответное сообщение на сообщение-запрос, которое его вызвало. При этом ответное сообщение должно содержать некоторый текст, описывающий характер ошибки, которая произошла при обработке запроса. Для более чистого и красивого кода будем обрабатывать произошедшие ошибки в `ExceptionMiddleware`, который перехватывает исключения, сравнивает их со списком известных ошибок и возвращает пользователю ответное сообщение с соответствующим этой ошибке текстом. При такой реализации архитектура бота выглядит проще, так как не нужно прокидывать много дополнительных параметров из `handlers` в `managers`, а затем и в `validators` слой.

Архитектура компонента нейросетевой модели является достаточно простой, так как представляет собой всего лишь из `event loop`, в котором для каждого нового поступившего сообщения загруженная модель выполняет генерацию. По мере выполнения также выполняется отправка сообщений в очередь “status” с процентной оценкой количества пройденных итераций генерации.

3.7 Тестирование системы

Тестирование системы проводилось вручную. Проверялась скорость работы системы, работа валидации корректности запросов, а также качество генерации, которое описано подробнее в пункте 3.5, посвященному проведению экспериментов.

3.7.1 Скорость работы

В среднем сервис обрабатывает один запрос за 23,6 секунды. Это время достигается при отсутствии ожидания запроса на генерацию в очереди, то есть в расчет включено только время принятия запроса от пользователя ботом, загрузка переданного изображения на сервер, обработка и валидация корректности запроса, постановка в очередь и получение сообщения без ожидания, генерация моделью итогового изображения и отправку ответа пользователю. Более подробно среднее время работы указано в таблице 1.

Таблица 1 – Среднее время генерации изображения для каждого компонента системы

	Валидация поступившего запроса	Ожидание в очереди	Генерация изображения	Отправка ответа
Время, сек	2,68	0	17,9	3,01

Распишем подробнее указанные в таблице 1 этапы генерации, время которых замерялось:

1. Валидация поступившего запроса. Сюда входит время на валидацию запроса, загрузку изображения скетча на сервер, сохранение новых данных в базу, а также помещение нового сообщения о генерации в очередь.

2. Ожидание в очереди. Так как в данной конфигурации имелся только 1 инстанс нейросетевой модели, то параллельная генерация несколькими моделями не поддерживается. Из-за этого все запросы обрабатываются моделью последовательно и в ситуации наличия нескольких одновременных запросов на генерацию образуется очередь, в которой время ожидания для рассматриваемого запроса рассчитывается как $t_{wait} = n * t_{processing} + t_{run}$, где n – количество сообщений в очереди “generation” перед рассматриваемым запросом, $t_{processing}$ – время генерации изображения (обработки одного запроса), t_{run} – время генерации для запроса, который был вытащен из очереди и обрабатывается в данный момент. При тестировании системы измерения проводились таким образом, что при попадании рассматриваемого запроса в очередь перед ним никогда не было ни одного другого запроса и модель в этот момент не обрабатывала запрос.

3. Генерация изображения. В данный этап входит перевод входных параметров в латентное пространство, вычисление первичного изображения, из которого будет происходить генерация, все итерации генерации, а также отправка сообщения о завершившейся обработке.

4. Отправка ответа. Этап включает в себя принятие ботом сообщения из очереди “status” о том, что генерация была завершена, загрузка итогового изображения, а также отправка нового сообщения пользователю с результатом генерации.

3.7.2 Обработка некорректных запросов

В процессе обработки поступившего запроса бот выполняет несколько проверок, чтобы отсеять заведомо некорректные запросы.

1. Запрос на генерацию должен иметь прикрепленное к нему изображение. Если изображение не было прикреплено, то такой запрос не будет обработан, так как даже не попадет в нужный handler.

2. Запрос на генерацию вместе с прикрепленным изображением должен иметь подпись к этому изображению, содержащую текстовое описание генерируемого контента (prompt). Если описание не было указано, то вернется ошибка «*No caption was found. Generation will not start*».

3. Соблюдается лимит на количество ожидающих результата принятых запросов на генерацию. Если лимит для пользователя превышен, вернется ошибка «*You have done too many requests. Please wait until some generation will be finished*». Значение лимита устанавливается в конфигурационном файле бота и по умолчанию равен 3.

4. В случае если не удалось загрузить переданное изображение на сервер, может вернуться одна из двух ошибок «*Image saving timeout*» или «*Image saving error*» - в зависимости от причины, по которой изображение не удалось загрузить.

5. При задании кастомных параметров генерации может вернуться ошибка «*Invalid parameter format*» в случае, если формат запроса составлен некорректно. В таблице 2 указаны несколько примеров правильно и неправильно оформленных запросов.

Таблица 2 – Примеры корректных и некорректных форматов запросов на изменение параметров генерации

Запрос	Запрос корректен
/grad_k 0.5	да
/grad_k param=0.5	нет
/grad_k пять десятых	нет
/grad_k0.5	нет
/grad_k 1/2	нет

6. Параметры генерации должны иметь значения из определенного интервала. В случае, если переданные значения не попадают в интервал, будет возвращена ошибка «*Invalid parameter value*».

3.8 Вывод

В процессе разработки и экспериментальных исследований была получена оптимальная архитектура модели для генерации изображения по текстовому описанию и переданному изображению скетча. После этого была реализована полноценная система для генерации, использующая в своей основе асинхронного Telegram бота и полученную нейросетевую модель. Система обладает хорошими показателями скорости работы и удовлетворяет всем сформулированным требованиям.

Заключение

В результате выполнения выпускной квалификационной работы была достигнута главная цель – создана система генерации изображений на основе эскизов и текстового описания. Система реализована в виде асинхронного Telegram бота.

В ходе работы выполнены следующие задачи:

1. выполнен обзор существующих методов генерации изображений на основе эскизов;
2. получены размеченные обучающие данные;
3. разработан метод генерации изображения;
4. экспериментально определена оптимальная архитектура сети и оптимальная реализация метода генерации;
5. система генерации реализована в виде полноценного сервиса;
6. проведено мануальное тестирование системы.

Сервис обладает хорошей скоростью работы и качеством генерации, а также удовлетворяет всем сформулированным требованиям. В дальнейшем для улучшения работы и повышения качества генерации следует продолжать экспериментальные исследования, направленные на поиск новых методов генерации и оценки сходства генерируемого изображения с входным изображением эскиза. При необходимости, можно добавить более детальную валидацию переданного изображения перед генерацией, а также валидацию текстового сообщения, что позволит увеличить надежность системы и предотвратить генерацию по заведомо некорректным входным данным.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Saul Dobilas. GANs: Generative Adversarial Networks — An Advanced Solution for Data Generation, 2022 URL: <https://towardsdatascience.com/gans-generative-adversarial-networks-an-advanced-solution-for-data-generation-2ac9756a8a99>
2. Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks, 2018 URL: <https://arxiv.org/abs/1611.07004>
3. Yongyi Lu, Shangzhe Wu, Yu-Wing Tai, Chi-Keung Tang. Image Generation from Sketch Constraint Using Contextual GAN, 2018 URL: <https://arxiv.org/abs/1711.08972>
4. Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics, 2015 URL: <https://arxiv.org/abs/1503.03585>
5. Jonathan Ho, Ajay Jain, Pieter Abbeel. Denoising Diffusion Probabilistic Models, 2020 URL: <https://arxiv.org/abs/2006.11239>
6. Alex Nichol, Prafulla Dhariwal. Improved Denoising Diffusion Probabilistic Models, 2021 URL: <https://arxiv.org/abs/2102.09672>
7. Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision, 2021 URL: <https://arxiv.org/abs/2103.00020>
8. Jonathan Whitaker. Mid-U Guidance: Fast Classifier Guidance for Latent Diffusion Models, 2023 URL: <https://wandb.ai/johnnowhitaker/miduguidance/reports/Mid-U-Guidance-Fast-Classifier-Guidance-for-Latent-Diffusion-Models--VmlldzozMjg0NzA1>
9. Jonathan Ho, Tim Salimans. Classifier-Free Diffusion Guidance, 2022 URL: <https://arxiv.org/abs/2207.12598>
10. Shin-I Cheng, Yu-Jie Chen, Wei-Chen Chiu, Hung-Yu Tseng, Hsin-Ying Lee. Adaptively-Realistic Image Generation from Stroke and Sketch with Diffusion Model, 2022 URL: <https://arxiv.org/abs/2208.12675>

11. Jooyoung Choi, Sungwon Kim, Yonghyun Jeong, Youngjune Gwon, Sungroh Yoon. ILVR: Conditioning Method for Denoising Diffusion Probabilistic Models, 2021 URL: <https://arxiv.org/abs/2108.02938>
12. Andrey Voynov, Kfir Aberman, Daniel Cohen-Or. Sketch-Guided Text-to-Image Diffusion Models, 2022 URL: <https://arxiv.org/abs/2211.13752v1>
13. Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models, 2022 URL: <https://huggingface.co/runwayml/stable-diffusion-v1-5>
14. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database URL: https://image-net.org/static_files/papers/imagenet_cvpr09.pdf
15. Zhuo Su, Wenzhe Liu, Zitong Yu, Dewen Hu, Qing Liao, Qi Tian, Matti Pietikäinen, Li Liu. Pixel Difference Networks for Efficient Edge Detection, 2021 URL: <https://arxiv.org/abs/2108.07009>
16. Chengying Gao, Qi Liu, Qi Xu, Limin Wang, Jianzhuang Liu, Changqing Zou. SketchyCOCO: Image Generation from Freehand Scene Sketches URL: <https://sysu-imsl.github.io/EdgeGAN/index.html>
17. Andrey Voynov, Kfir Aberman, Daniel Cohen-Or. Sketch-Guided Text-to-Image Diffusion Models, 2022 URL: <https://arxiv.org/abs/2211.13752v1>
18. Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, Stefano Ermon. SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations, 2022 URL: <https://arxiv.org/abs/2108.01073>
19. Stack Overflow, 2022 URL: <https://ru.stackoverflow.com/questions/1275801/telebot-%D0%BF%D1%80%D0%BE%D1%82%D0%B8%D0%B2-aiogram>