СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние годы нейронные сети и технологии компьютерного зрения стали важными инструментами для работы с визуальной информацией. Одна из актуальных задач, требующая решения, – это автоматизация процесса создания видео на основе изображений. Такая задача находит применение в самых различных сферах, включая медицину, кинематограф и другие сферы деятельности, где анимация статичных изображений может быть трудоемким и длительным процессом.

Генеративно-состязательные сети (GAN) представляют собой мощный инструмент для решения задачи создания видеопоследовательностей на основе изображений. Они способны генерировать новые кадры, основываясь на исходных данных, таких как статичные изображения, и создавать реалистичные видеопоследовательности. Этот подход может существенно упростить создание анимации и сэкономить ресурсы на ручное создание видеоматериалов.

Цель данного дипломного проекта – разработка программного средства, которое позволит автоматически создавать анимацию из изображений с использованием генеративно-состязательных нейронных сетей. Это программное решение ориентировано на специалистов, работающих с визуальными эффектами и анимацией, а также может быть полезно в образовательных или научных проектах.

Для достижения поставленной цели необходимо решить следующие задачи:

- исследовать и сравнить существующие архитектуры нейронных сетей для генерации видео на основе изображений;
- выбрать подходящую модель генеративно-состязательной сети для реализации проекта;
- подготовить набор данных для обучения, включающий видеопоследовательности и изображения;
- обучить и протестировать модель на реальных данных;
- разработать программное средство с удобным интерфейсом для работы с пользователями;
- оценить качество работы модели и протестировать программное средство.

Реализация данного проекта позволит упростить создание анимации, сделав этот процесс более доступным и менее затратным по времени и сложности для специалистов, работающих с мультимедийными данными.

Данный дипломный проект выполнен мной лично, проверен на заимствования, процент оригинальности составляет XX% (отчет о проверке на заимствования прилагается).

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Аналоги программ для генерации видео

В последние годы наблюдается бурное развитие технологий в области анимации изображений, что позволило создать ряд программных средств, использующих искусственный интеллект и нейронные сети для автоматической генерации анимаций из статичных изображений. Эти инструменты находят применение в различных областях, таких как создание мультфильмов, видеоигр, рекламных роликов, а также в медицине и научных исследованиях.

1.1.1 Lumen5 [1] – это платформа для создания видео, разработанная для брендов и предприятий, позволяющая создавать увлекательный видеоконтент для социальных постов, историй и рекламы.

Их цель – дать возможность любому человеку без обучения или опыта легко создавать потрясающие видео за считанные минуты.

Эта технология позволяет маркетинговым командам сосредоточиться на истории и повествовании, полагаясь на данную систему для выполнения тяжелой работы.

Платформа не только предлагает инструменты для создания видео, Lumen5 делает еще один шаг вперед, предоставляя все ресурсы, необходимые для создания захватывающих видеороликов.

Встроенная медиабиблиотека предлагает доступ к миллионам стоковых видеоматериалов, фотографий и музыкальных треков.

Это означает, что наши пользователи имеют доступ ко всему, что им нужно, без необходимости записывать или приобретать какие-либо внешние цифровые ресурсы.

Видеомонтаж – это трудоемкий процесс, поскольку он требует множества мелких правок и тонкой настройки. Используя машинное обучение, Lumen5 автоматизирует эти задачи, чтобы помочь пользователям создавать качественные видео с минимальными затратами времени, усилий и обучения. Например, длительность каждой сцены алгоритмически определяется на основе количества слов в сцене, текст размещается разумно, чтобы не закрывать важные темы, такие как человеческие лица, важные ключевые слова автоматически подчеркиваются и выделяются цветом бренда, а в конце каждого видео генерируется титровая сцена, чтобы помочь соблюдать законы об авторских правах.

1.1.2 Synthesia [2] – это компания, занимающаяся синтетическим медиапроизводством, которая разрабатывает программное обеспечение для создания видеоконтента, сгенерированного с помощью ИИ. По состоянию на январь 2025 года ее клиентская база включает более шестидесяти процентов компаний из списка Fortune 100. Она базируется в Лондоне, Англия.

Synthesia чаще всего используется корпорациями для общения,

ориентации и обучающих видеороликов. Она использовалась в рекламных кампаниях, отчетах, демонстрациях продуктов и для создания чат-ботов.

Программный алгоритм Synthesia имитирует речь и движения лица на основе видеозаписей речи человека и произношения фонем. Из этого создается видео с преобразованием текста в речь, которое выглядит и звучит как человек.

Пользователи создают контент с помощью предварительно сгенерированных ИИ-презентаторов платформы или путем создания цифровых представлений самих себя или личных аватаров с помощью инструмента для редактирования видео с использованием ИИ на платформе. Эти аватары могут использоваться для озвучивания видеороликов, сгенерированных из текста. По состоянию на август 2021 года база данных голосов Synthesia включала несколько вариантов пола на более чем шестидесяти языках.

Платформа запрещает использование своего программного обеспечения для создания несогласованных клонов, в том числе знаменитостей или политических деятелей в сатирических целях. Для использования образа человека необходимо предоставить явное согласие в дополнение к строгому режиму предварительной проверки, чтобы избежать «дипфейка».

1.1.3 InVideo [3] – это компания, которая предлагает инструменты для создания видео на основе искусственного интеллекта, помогающие пользователям превращать свои идеи в увлекательные видеоролики.

Их флагманский продукт InVideo AI позволяет пользователям создавать видео с использованием более 5000 готовых шаблонов, доступных на их платформе. С помощью InVideo AI Studio пользователи могут ввести любую тему и позволить искусственному интеллекту сгенерировать сценарий, создать сцены, добавить закадровый голос и настроить видео в соответствии со своими предпочтениями. Платформа также предлагает полный контроль над редактированием, позволяя пользователям легко вносить изменения и корректировать созданные видео.

InVideo обслуживает широкий спектр коммерческих рынков с клиентской базой более 7 миллионов человек в 190 странах. Их продукт получил признание благодаря высоким рейтингам, что делает его популярным выбором для частных лиц и предприятий, желающих эффективно создавать профессиональные видеоролики.

Инструменты искусственного интеллекта InVideo охватывают различные аспекты создания видео, включая редактирование, текстовые подсказки, переходы, закадровый голос и функции совместной работы для редактирования в реальном времени. Компания стремится помочь брендам расширить свое присутствие в Интернете с помощью стратегий видеоконтента, успеха в социальных сетях и возможностей монетизации. InVideo предлагает гибкие тарифные планы, включая ежемесячные и годовые варианты со скидкой 20%. Они также предоставляют мобильное приложение для создания видео на ходу и круглосуточную человеческую поддержку для

любой необходимой помощи.

1.2 Способы генерации видео

Генерация видео – это сложная задача, требующая синтеза последовательностей изображений (кадров), которые связаны между собой временной и пространственной согласованностью. На сегодняшний день существует несколько основных подходов и методов для генерации видео, включая классические алгоритмы и методы глубокого обучения.

Глубокие генеративные модели применяются в различных областях, таких как синтез изображений, аудио, видео и обработка естественного языка. С быстрым развитием методов глубокого обучения в последние годы произошел взрыв различных глубоких генеративных моделей. Это привело к растущему интересу к сравнению и оценке этих моделей с точки зрения их производительности и применимости к различным областям.

Далее представлено всестороннее сравнение глубоких генеративных моделей, включая диффузионные модели, генеративно-состязательные сети и вариационные автокодировщики (VAE) (рисунок 1.1).

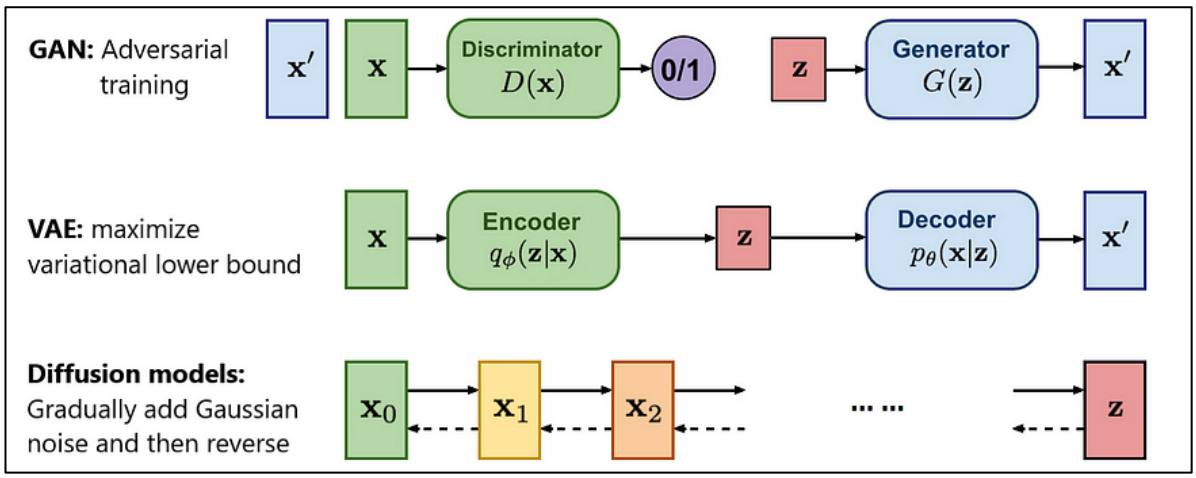

Рисунок 1.1 – Обзор различных типов генеративных моделей [4]

1.2.1 Генеративно-состязательные сети были предложены Иэном Гудфеллоу в 2014 году [5] и стали одним из самых значительных достижений в области глубокого обучения.

GAN учатся генерировать новые данные, схожие с обучающим набором данных. Она состоит из двух нейронных сетей – генератора и дискриминатора, которые «играют» в двустороннюю игру.

Генератор получает на вход случайные значения, взятые из нормального распределения, и создает синтетическую выборку, в то время как дискриминатор пытается различить реальный и сгенерированный образец. Генератор обучается выдавать реалистичный результат, который может обмануть дискриминатор, в то время как дискриминатор, если обучен правильно, пытается отличать реальные данные от сгенерированных.

Цель состоит в том, чтобы сделать дискриминатор неспособным различать истинные и сгенерированные образцы.

Состязательная функция потерь не мотивирует генератор покрывать все распределение данных. Когда дискриминатор переобучен или происходит катастрофическое забывание, генератор может быть достаточно «удовлетворен» созданием ограниченного набора образцов с низким разнообразием. Это распространенная проблема, которая называется коллапсом мод.

В идеале нейронная сеть сходится и дискриминатор не может отличить реальные образцы от сгенерированных. Это приводит к очень реалистичным образцам. С другой стороны, определить момент сходимости сети бывает сложно. Вместо того чтобы отслеживать снижение одной функции потерь, нужно следить за двумя, которые не всегда имеют простую интерпретацию. Иногда непонятно, что происходит с нейронной сетью.

Несмотря на эти вызовы, GAN остаются одной из самых популярных и перспективных технологий в области генерации данных. Их успешное применение в задачах, связанных с визуальными и мультимедийными данными, делает их особенно ценными для создания анимаций и видеопоследовательностей, что и является основной задачей данного дипломного проекта.

Верхний ряд рисунка 1.1 показывает схему работы GAN.

1.2.2 VAE [4, 6] состоят из кодера и декодера. Кодер отображает высокоразмерные входные данные в низкоразмерное представление, в то время как декодер пытается восстановить исходные высокоразмерные входные данные, отображая это представление обратно в их исходную форму. Кодер выводит нормальное распределение латентного кода как низкоразмерное представление, предсказывая векторы среднего и стандартного отклонения.

Обучение проводится путем максимизации логарифма правдоподобия, который после математических упрощений сводится к функции потерь L2 (MSE). Оценивается расхождение между входными и сгенерированными образцами.

На выходе обычно получаются образцы с низкой реалистичностью. На это есть несколько причин:

1 Поскольку кодер предсказывает распределение латентного кода, могут быть случаи, когда два распределения латентных кодов перекрываются. Поэтому, если два входных изображения имеют одинаковый латентный код, оптимальной декодировкой будет усреднение этих двух входных изображений. Это приводит к размытым образцам. У GAN и диффузионных моделей такой проблемы нет.

2 Используется пиксельная функция потерь. Например, генерация изображения с волосами будет состоять из чередующихся светлых и темных пикселей. Если сдвинуть сгенерированное изображение всего на один пиксель, функция потерь по сравнению с исходным изображением значительно

увеличится или уменьшится. Однако VAE не сохраняют такую пиксельную информацию, потому что латентное пространство значительно меньше изображения. Это заставляет модель предсказывать усредненные светлые и темные пиксели для нахождения оптимального решения, что приводит к размытым изображениям. У GAN такой проблемы нет, потому что дискриминатор может использовать размытость образцов для различения реальных и сгенерированных изображений. Аналогично, диффузионные модели, несмотря на наличие пиксельной функции потерь, не имеют этой проблемы. Они опираются на текущую шумную структуру изображения, полученную из исходных данных, чтобы предсказать следующий шаг шумоподавления.

Максимизация правдоподобия заставляет покрывать все режимы обучающего набора данных, что дает нейросети возможность обрабатывать каждый обучающий пример. Получаются образцы с высоким разнообразием.

VAE легка в обучении, так как имеется всего одна вычислимая функция правдоподобия. Средний ряд рисунка 1.1 демонстрирует его работу.

1.2.3 Диффузионные модели [4, 7] состоят из процессов прямой и обратной диффузий.

Прямая диффузия – это многошаговый процесс, который постепенно добавляет шум к входным данным до тех пор, пока не будет получен белый шум. Это необучаемый процесс, который обычно занимает 1000 шагов.

Процесс обратной диффузии направлен на то, чтобы пошагово обратить прямой процесс, удаляя шум для восстановления исходных данных. Процесс обратной диффузии реализуется с использованием обучаемой нейронной сети.

Здесь обучение происходит, как и в случае с VAE, за счет максимизации логарифма правдоподобия, который после математических упрощений сводится к функции потерь L2. Во время обучения вычисляются зашумленные изображения для шагов N и N-1, используя формулу для случайно выбранного значения N. Диффузионная модель затем предсказывает изображение на шаге N-1, основываясь на зашумленном изображении на шаге N. Сгенерированное изображение сравнивается с изображением на шаге N-1 с использованием функции потерь L2.

Диффузионные модели способный генерировать высококачественные образцы. Это связано с природой поэтапного удаления шума. В отличие от VAE и GAN, которые создают образцы сразу, диффузионные модели генерируют их шаг за шагом. Сначала модель создает грубую структуру изображения, а затем добавляет на нее мелкие детали.

Также эта модель выдает высокое разнообразие генерируемых образцов и является легкой в обучении.

В отличие от GAN и VAE, требуется многократный запуск нейронной сети для поэтапной генерации образцов. Хотя существуют методы выборки, которые могут значительно ускорить этот процесс, они все же медленнее, чем у GAN и VAE.

На рисунке 1.2 показаны сильные и слабые стороны описанных генеративных моделей.

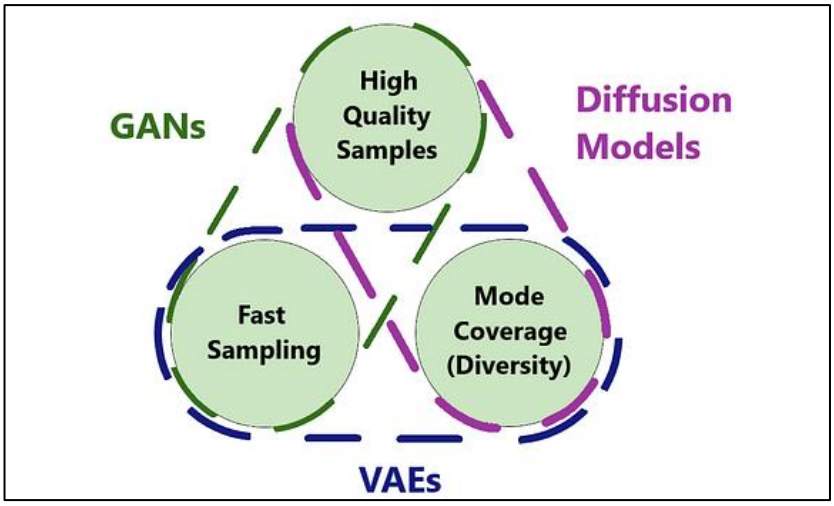

Рисунок 1.2 – Сильные и слабые стороны генеративных моделей [4]

1.2.4 Кроме описанных методов создания видео на основе генеративных моделей есть ещё и другие. Например:

1 Рекуррентные нейронные сети (RNN) [8] и их модификации, такие как LSTM (англ. Long Short-Term Memory) и GRU (англ. Gated Recurrent Unit), могут предсказывать следующее изображение в видеопоследовательности, учитывая предыдущие кадры. Этот способ подходит для генерации временных данных, однако может сталкиваться с проблемами при создании длинных и сложных видеопоследовательностей.

2 Методы на основе 3D-анимации (например, NeRF, Neural Radiance Fields [9]) позволяют создавать реалистичные видеопоследовательности с трехмерными сценами, которые могут изменять ракурс или движение объектов в пространстве. Эти методы активно развиваются и применяются в генерации сложных видеопоследовательностей с изменением перспективы.

Каждый из перечисленных способов имеет свои преимущества и недостатки, и выбор метода зависит от целей проекта, требуемого уровня реалистичности и вычислительных ресурсов.

1.3 Типы и модификации GAN

С момента появления оригинальной архитектуры GAN было разработано множество улучшений и вариантов, каждый из которых решает специфические задачи или преодолевает ограничения базовой модели.

1.3.1 Синтез высококачественных изображений из текстовых описаний является сложной проблемой в компьютерном зрении и имеет множество практических приложений. Образцы, сгенерированные существующими подходами преобразования текста в изображение, могут приблизительно

отражать смысл заданных описаний, но они не содержат необходимых деталей и ярких частей объекта.

В 2017 году была предложена архитектура под названием Stacked Generative Adversarial Networks (StackGAN) [10] для генерации фотореалистичных изображений размером 256x256, обусловленных текстовыми описаниями. В ней сложная задача раскладывается на более простые, управляемые подзадачи с помощью процесса уточнения эскиза.

На первом этапе (Stage-I) GAN рисует примитивную форму и цвета объекта на основе заданного текстового описания, получая изображения с низким разрешением Stage-I.

На втором этап (Stage-II) GAN принимает результаты Stage-I и текстовые описания в качестве входных данных и генерирует изображения с высоким разрешением с фотореалистичными деталями. Она способна исправлять дефекты в результатах Stage-I и добавлять убедительные детали с помощью процесса уточнения.

На рисунке 1.3 приведена структура StackGAN.

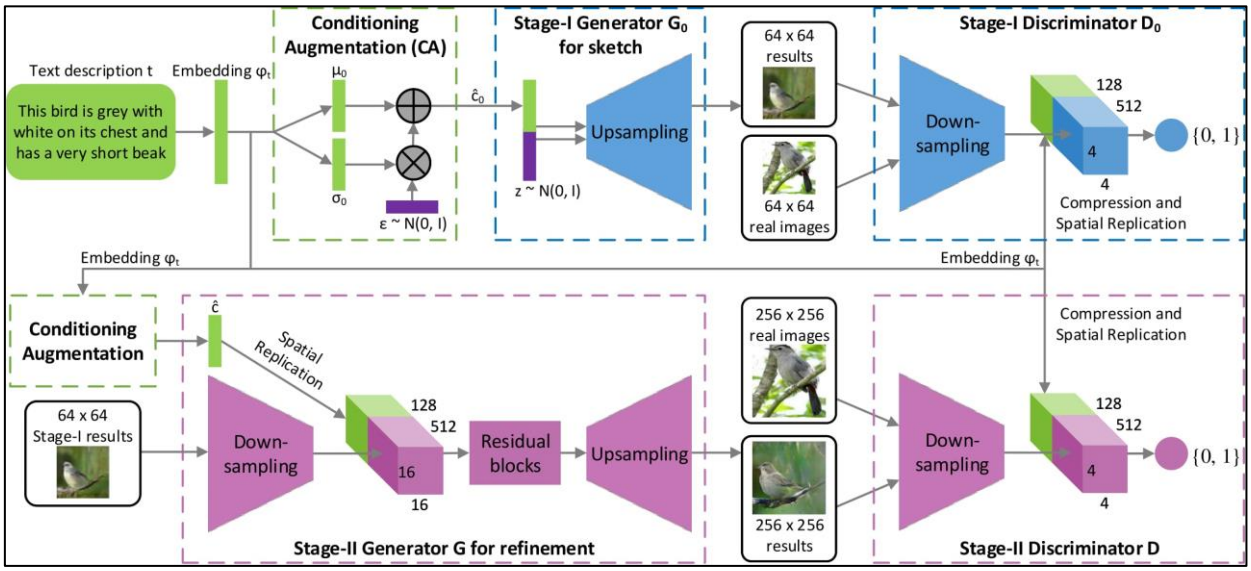

Рисунок 1.3 – Структура StackGAN [10]

Для улучшения разнообразия синтезированных изображений и стабилизации обучения условной GAN была введена новая техника Conditioning Augmentation, которая способствует сглаживанию в латентном условном многообразии. Обширные эксперименты и сравнения с современными данными на эталонных наборах данных показывают, что предлагаемый метод достигает значительных улучшений в создании фотореалистичных изображений, обусловленных текстовыми описаниями.

1.3.2 DVD-GAN (англ. Dual Video Discriminator GAN) [11] представляет собой мощную архитектуру, разработанную для генерации видеопоследовательностей высокого разрешения. В её основе лежит использование двух дискриминаторов: один анализирует качество отдельных

кадров, а другой – временную согласованность видео.

Такой подход позволяет лучше учитывать как пространственные, так и временные зависимости, что способствует созданию более реалистичных и согласованных видеопоследовательностей.

Структура DVD-GAN показана на рисунке 1.4.

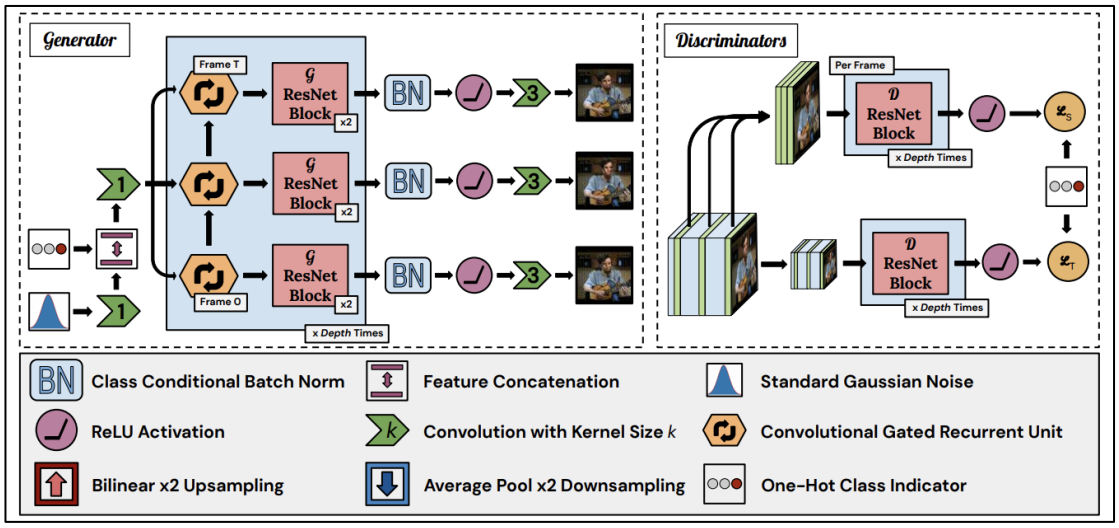

Рисунок 1.4 – Структура DVD-GAN [11]

1.3.3 TGAN (англ. Temporal GAN) [12], в свою очередь, делает акцент на временной согласованности генерируемого контента.

Этот подход делит процесс на два этапа: сначала генерируются ключевые кадры, а затем создаются промежуточные кадры, которые обеспечивают плавность и непрерывность движения.

Модель подходит для задач, где важно поддерживать качество динамики и временную структуру видео.

Структура TGAN показана на рисунке 1.5.

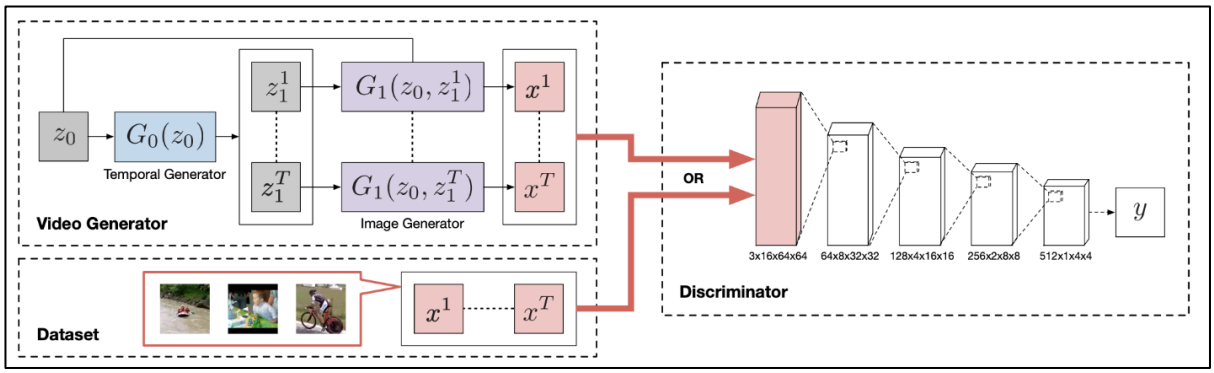

Рисунок 1.5 – Структура TGAN [12]

1.3.4 VideoGAN [13] – одна из первых моделей, адаптированных для генерации видео, в которой стандартная архитектура GAN была расширена для работы с временными данными.

Хотя эта модель является базовой для задач генерации видео, она может сталкиваться с трудностями в создании длительных стабильных видеопоследовательностей, что требует дальнейших улучшений в управлении движением.

На рисунке 1.6 показана структура такой сети.

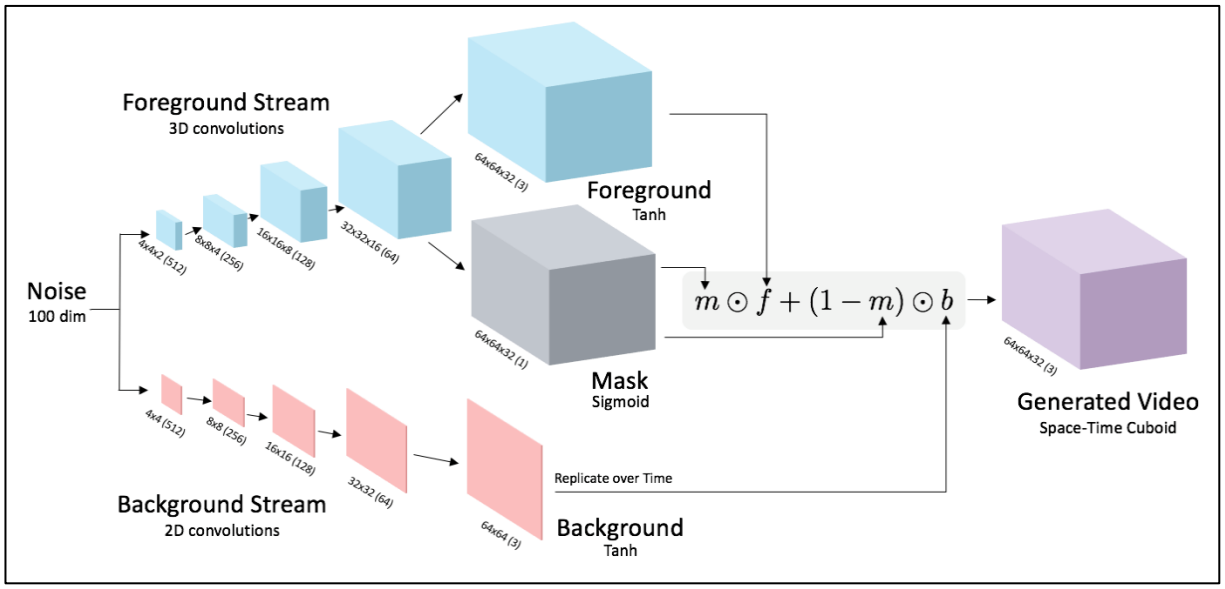

Рисунок 1.6 – Структура VideoGAN [13]

1.3.5 MoCoGAN (англ. Motion and Content GAN) [14] отличается тем, что разделяет задачу генерации видео на два компонента: один отвечает за статический контент изображения, а другой — за его динамику. Благодаря этому модель может более эффективно синхронизировать и управлять движением в сгенерированных видеороликах, что способствует созданию реалистичных анимаций с сохранением качественного статического контента.

Интересным решением является TiVGAN (англ. Text-to-Image-to-Video GAN) [15], который использует двухшаговую стратегию генерации. Сначала из текстового описания создается изображение, которое затем превращается в видеопоследовательность. Это делает модель гибкой для задач, где требуется генерировать видео на основе текстовых данных, например, для синтеза анимации из текстовых описаний.

Для прогнозирования будущих кадров на основе текущих изображений применяется модель FutureGAN [16], которая позволяет предсказывать, как изменится видеопоследовательность со временем. Это особенно полезно в задачах прогнозирования движения объектов и предсказания будущих событий на основе предыдущих данных.

Таким образом, каждая из этих архитектур предлагает уникальный подход к решению задачи генерации видео, используя разные способы работы с временными и пространственными данными, обеспечивая гибкость и улучшение качества сгенерированного контента.

1.4 Python

Python [17] – один из самых популярных языков программирования [18]. Он был создан Гвидо ван Россумом и выпущен в 1991 году. Он имеет эффективные высокоуровневые структуры данных и простой, но действенный подход к объектно-ориентированному программированию. Элегантный синтаксис и динамическая типизация Python, вместе с его интерпретируемой природой, делают его идеальным языком для написания скриптов и быстрой разработки приложений во многих областях на большинстве платформ.

Этот язык считается довольно полезным и удобным для работы с искусственным интеллектом (ИИ), и по статистике это третий по популярности язык после JavaScript и HTML/CSS. Индекс TIOBE, который учитывает популярность поисковых запросов в рейтинге, в настоящее время ставит Python на первое место [19]. Сегодня на Python работают множество известных организаций, например, Google, Facebook, Instagram, Spotify, Netflix, Quora.

Его мощь проявляется в способности автоматизировать задачи и оптимизировать рабочие процессы. Удобный синтаксис Python делает его более привлекательным для программистов по сравнению с другими языками, что в конечном итоге способствует повышению эффективности разработчиков. Также сегодня на GitHub есть более миллиона репозиториев. В результате все эти факторы делают Python одним из наиболее предпочтительных языков программирования.

Кроме того, Python используется для веб-разработки (серверной), разработки ПО, математики, системных скриптов.

1.5 FastAPI

FastAPI [20] – это современный, быстрый и высокопроизводительный веб-фреймворк для создания API используя Python.

Фреймворк полностью поддерживает асинхронное программирование, позволяя писать асинхронные обработчики маршрутов и использовать преимущества синтаксиса `async/await` в Python для неблокирующих операций ввода-вывода.

Имеется автоматическая генерация интерактивной и удобной для пользователя документации по API. Используются стандарты OpenAPI и JSON Schema для предоставления исчерпывающей документации для вашего API, включая проверку входных данных, ожидаемые ответы и многое другое. Существует несколько вариантов документирования, два из которых включены по умолчанию: Swagger UI и ReDoc.

Благодаря интеграции FastAPI с Pydantic можно определять модели данных с помощью подсказок типов (type hints) Python, обеспечивая автоматическую проверку данных и сериализацию. Эта функция помогает выявлять ошибки на ранних стадиях процесса разработки и повышает читаемость кода.

FastAPI поддерживает внедрение зависимостей, позволяя эффективно управлять зависимостями и организовывать их. Эта функция особенно полезна при работе с подключениями к базе данных, аутентификацией и другими общими ресурсами.

Синтаксис FastAPI понятен, легок для чтения и очень похож на стандартное определение функций Python, что делает его доступным как для начинающих, так и для опытных разработчиков.

У FastAPI были (и есть) куча альтернатив:

1 Flask – легкий и широко используемый веб-фреймворк в экосистеме Python. Он прост в использовании и с ним легко начать работу, но ему не хватает встроенной поддержки асинхронности и автоматической проверки на основе подсказок типов.

2 Django – полнофункциональный веб-фреймворк, который следует философии “батарейки включены”. Он предоставляет множество готовых функциональных возможностей, включая ORM, интерфейс администратора и многое другое, но для небольших и простых API это может оказаться излишним. В нем зашит паттерн MVC. По сути это модульный монолит, который не такой гибкий, как FastAPI.

В целом это два самые популярные фреймворки, которые носили по-настоящему массовый характер.

Описанные ниже фреймворки были и есть не очень популярны, не очень удобны, либо не очень производительны:

1 Tornado – асинхронный веб-фреймворк, который может обрабатывать большое количество одновременных подключений. Он часто используется в сценариях, требующих высокого уровня параллелизма, но может иметь более крутую кривую обучения по сравнению с FastAPI (тяжел, местами не очевиден, не очень удобен).

2 Bottle – минималистичный веб-фреймворк, разработанный для маломасштабных приложений. Он легкий и простой в использовании, но ему не хватает скорости, масштабируемости и оптимизации производительности, присутствующих в FastAPI.

Таким образом, можно перечислить основные достоинства FastAPI:

- скорость;
- встроенная поддержка проверки типов;
- возможность реализации высоконагруженных API за счет асинхронности.

1.6 Способы представления результатов

Для демонстрации результатов работы модели, генерирующей видео, необходимо учитывать несколько аспектов:

- удобство использования;
- интерактивность;
- доступность;
- визуальная привлекательность.

Способы представления результатов могут включать создание различных интерфейсов и платформ, которые позволяют пользователю легко загружать входные изображения, просматривать анимации, а также управлять параметрами генерации.

Тенденция науки о данных и аналитики растет с каждым днем. Из конвейера науки о данных одним из самых важных шагов является развертывание модели.

1.6.1 Веб-приложения являются удобным способом представления результатов, позволяющим пользователю получать доступ к системе через любой браузер без необходимости установки дополнительного ПО. Веб-приложения могут быть более сложными и функциональными, чем простые интерфейсы, и предлагают больше возможностей для визуализации и настройки.

Существует множество вариантов в Python для развертывания модели. Так, на сегодняшний день есть несколько популярных веб-фреймворков – Django, FastAPI, Flask. Но проблема с их использованием заключается в том, что для красивого визуального и удобного взаимодействия пользователя с моделью должна быть создана некоторая Frontend-часть и, очевидно, иметься некоторые знания об HTML, CSS и JavaScript. Учитывая эти предварительные условия, Адриен Трей, Тиаго Тейшейра и Аманда Келли создали «Streamlit».

Streamlit [21] – это простая в использовании библиотека Python для создания веб-приложений. Она позволяет быстро разрабатывать интерфейсы для визуализации и взаимодействия с нейросетевыми моделями. Преимуществом Streamlit является его простота в реализации: достаточно нескольких строк кода, чтобы создать веб-интерфейс для загрузки изображений, демонстрации генерируемых видео и настройки параметров. Это идеально подходит для демонстрации работы модели генерации анимаций.

С его помощью можно легко развернуть любую модель машинного обучения и любой проект Python, не беспокоясь о frontend части. Streamlit очень удобен для пользователя. Также преимуществом этой библиотеки является то, что одноименная компания предоставляет бесплатную возможность развертывания приложения, написанного с помощью этой библиотеки, на серверах компании.

1.6.2 Для более мощных решений, а также с целью увеличения доступности системы можно интегрировать модель с облачными сервисами. Эти платформы позволяют работать с большими объемами данных и мощными вычислительными ресурсами, что важно для эффективной генерации видео с использованием GAN.

Google Colab [22] – это платформа для работы с Jupyter-ноутбуками, предоставляющая бесплатные вычислительные ресурсы, включая графические процессоры (GPU). Google Colab является отличным вариантом для демонстрации и тестирования моделей GAN, так как позволяет запускать

модели в облаке, использовать GPU для ускорения процесса генерации и легко делиться результатами с другими пользователями.

Google Colab устраняет необходимость в сложной настройке конфигурации и установке, поскольку работает прямо в браузере. Он также включает в себя предустановленные библиотеки Python, которые не требуют настройки для использования.

Jupyter Notebook – это бесплатное творение с открытым исходным кодом из проекта Jupyter. Jupyter Notebook похож на интерактивный лабораторный блокнот, который включает в себя не только заметки и данные, но и код, который может манипулировать данными. Код может быть выполнен в блокноте, который, в свою очередь, может захватывать вывод кода. Такие приложения, как Matlab и Mathematica, стали пионерами этой модели, но, в отличие от этих приложений, Jupyter – это веб-приложение на основе браузера.

Google Colab построен на основе кода Project Jupyter и размещает блокноты Jupyter без необходимости установки локального программного обеспечения. Но в то время, как блокноты Jupyter поддерживают несколько языков, включая Python, Julia и R, Colab в настоящее время поддерживает только Python.

Блокноты Colab хранятся в учетной записи Google Drive и могут совместно использоваться с другими пользователями, как и другие файлы Google Drive. Блокноты также включают функцию автоматического сохранения, но они не поддерживают одновременное редактирование, поэтому совместная работа должна быть последовательной, а не параллельной.

Colab бесплатен, но имеет ограничения. Некоторые типы кода запрещены, например, обслуживание мультимедиа и майнинг криптовалют. Доступные ресурсы также ограничены и зависят от спроса, хотя Google Colab предлагает профессиональную версию с более надежными ресурсами. Существуют и другие облачные сервисы на основе Jupyter Notebook, включая Azure Notebooks от Microsoft и SageMaker Notebooks от Amazon.

Для повышения интерактивности и вовлеченности пользователей можно использовать различные инструменты для демонстрации анимаций в реальном времени. К примеру, можно внедрить модель генерации видео в какую-либо социальную сеть.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Процесс проектирования системы для генерации видео включает несколько ключевых компонентов, каждый из которых выполняет свою роль в обеспечении качественного результата. Проект включает две нейронные сети: одну для генерации изображений на основе текста и вторую – для создания видео на основе сгенерированных изображений.

Было выделено несколько блоков, каждый из которых отвечает за определенный этап обработки данных:

- блок сервисов;
- блок логирования;
- блок конфигурации;
- блок модели;
- блок загрузки/выгрузки модели;
- блок обучения модели;
- блок данных для обучения;
- блок пользовательского интерфейса.

Это структурное разделение позволяет эффективно управлять процессом генерации, а также улучшать каждый компонент системы по мере необходимости.

Взаимосвязь между основными компонентами проекта отражена на структурной схеме ГУИР.400201.042 С1.

2.1 Блок сервисов

Блок сервисов в проектируемой системе отвечает за организацию логики приложения и управление взаимодействием между различными компонентами системы. Архитектура проекта основана на принципах раздельного управления логикой и обработкой данных, однако она не использует полный паттерн MVC (англ. Model-View-Controller). Вместо этого фокус делается на создании API, которое будет взаимодействовать с остальными модулями системы, такими как модели нейронных сетей, обучение и обработка данных.

Приложение разделено на несколько независимых сервисов, каждый из которых выполняет свою строго определенную задачу. Это обеспечивает модульность и гибкость, что упрощает разработку, сопровождение и масштабирование системы. Каждый сервис управляет своей областью ответственности, взаимодействуя с остальными модулями системы через интерфейсы и API.

Основная бизнес-логика приложения реализована на уровне сервисов, которые принимают входные данные, обрабатывают их и инициируют выполнение ключевых операций, таких как генерация изображений на основе текста и создание видео. Эти сервисы взаимодействуют с другими блоками системы, такими как модели нейронных сетей, модулем логирования и конфигурации.

Контроллеры в данной архитектуре играют роль основных интерфейсов для взаимодействия с клиентами и другими частями системы через API. Ответы от сервисов возвращаются обратно пользователю или другим модулям через контроллеры. Это позволяет обеспечить четкую последовательность действий и изолировать внутреннюю логику приложения от внешнего взаимодействия.

Каждый из сервисов взаимодействует с другими компонентами через четко определенные интерфейсы. Такой подход делает возможным модульное изменение или улучшение отдельных компонентов системы без необходимости изменения всего приложения.

2.2 Блок логирования

Блок логирования в проектируемой системе отвечает за сбор, хранение и обработку информации о событиях, происходящих в процессе работы приложения. Логирование является важной частью системы, так как оно помогает отслеживать выполнение операций, выявлять ошибки и проблемные участки кода, а также мониторить состояние приложения в реальном времени.

Это особенно важно в системах с машинным обучением, так как процесс обучения моделей и генерации изображений/видео может быть достаточно сложным и требовать детального мониторинга.

В системе логируются несколько ключевых типов данных:

1 Запросы и ответы API. Каждый запрос к приложению, будь то запрос на генерацию изображений или обучение модели, логируется вместе с информацией о запросе и ответе. Это позволяет легко отслеживать работу API и взаимодействие между клиентом и сервером.

2 Ошибки и исключения. Важной частью логирования является сбор информации об ошибках и исключениях, возникающих в ходе работы программы. Это необходимо для оперативного устранения проблем и улучшения надежности приложения.

3 Процесс обучения модели. Информация о ходе обучения нейронных сетей также сохраняется. Включаются такие данные, как число эпох, значения потерь при обучении, параметры оптимизатора и любые отклонения в процессе обучения.

Также в системе могут использоваться несколько уровней логирования, чтобы обеспечить баланс между детализацией информации и производительностью:

- DEBUG;
- INFO;
- WARNING;
- ERROR;
- CRITICAL.

Для хранения логов могут использоваться различные подходы, начиная от обычного логирования в текстовые файлы и заканчивая использованием специализированных систем для хранения и анализа логов, таких как

Elasticsearch, Graylog или Fluentd. Выбор системы хранения зависит от требований к производительности и объема данных.

Локальные файлы – наиболее простой способ хранения логов, при котором вся информация записывается в файлы на диске. Данный метод удобен для начального развертывания приложения, но может оказаться недостаточным в масштабируемых системах с большим объемом данных.

При развертывании системы в распределенной среде логирование может происходить в централизованное хранилище. Это позволяет собирать логи с различных сервисов в одном месте и эффективно анализировать их с помощью специализированных инструментов.

Для эффективного управления логами может применяться система обработки и анализа логов. Это позволяет не только хранить данные, но и автоматически выявлять ошибки, аномалии или неожиданные изменения в поведении системы. Возможна интеграция с системами мониторинга, которые будут уведомлять разработчиков о проблемах в реальном времени.

2.3 Блок конфигурации

Основная цель блока конфигурации – предоставить единое место для управления настройками приложения. Это позволяет не только гибко менять параметры без необходимости вносить изменения в код, но и упростить процесс развертывания системы на разных средах (например, на этапе разработки и в продакшн-среде). Правильно организованный блок конфигурации способствует повышению гибкости и масштабируемости системы.

Конфигурационные данные могут быть сохранены в одном или нескольких файлах. Наиболее часто используемые форматы файлов конфигурации:

- YAML – широко используемый формат для структурированных данных, удобный для чтения и редактирования человеком;
- JSON – ещё один популярный формат, который часто используется для конфигурационных файлов благодаря его простоте и возможности легко интегрировать с различными библиотеками и фреймворками;
- INI или TOML – текстовые форматы, подходящие для менее сложных настроек, также могут использоваться в зависимости от требований к проекту;
- .env-файлы – используются для хранения чувствительной информации, такой как ключи API, доступы к базам данных и параметры окружения.

Блок конфигурации содержит различные параметры, которые определяют поведение системы на разных этапах работы. Эти параметры можно разделить на несколько категорий:

- параметры обучения моделей;
- параметры оптимизаторов;
- тип генератора и дискриминатора;
- параметры данных;
- параметры генерации;

- параметры логирования и отладки;
- параметры внешних сервисов.

На этапе инициализации системы происходит загрузка конфигурационных файлов. Важно, чтобы перед запуском всех процессов происходила проверка корректности конфигурации (валидация). Это гарантирует, что все необходимые параметры заданы корректно, и исключает возможные ошибки, связанные с некорректной настройкой системы. Для валидации могут использоваться библиотеки, которые проверяют типы данных, диапазоны значений и наличие всех обязательных параметров.

2.4 Блок модели

Блок модели является центральным компонентом системы, который отвечает за использование и управление нейронными сетями для генерации контента. Он выполняет ключевую роль в проекте, обеспечивая функциональность для использования предобученных весов моделей в соответствии с требованиями проекта.

Одна из ключевых функций блока модели – генерация изображений на основе входных текстовых данных. Модель получает текстовое описание и преобразует его в изображение, использующее генеративные подходы. В данном контексте блок модели может содержать несколько подмодулей, отвечающих за создание изображений на разных этапах генерации:

Вторая основная задача блока модели – создание видео на основе сгенерированных изображений. Это может включать как создание последовательности кадров, так и более сложные задачи, такие как генерация видео с движением или с учетом временных зависимостей между кадрами.

Данный блок тесно взаимодействует с другими компонентами системы. Блок сервисов вызывает функции блока модели, передавая соответствующие параметры для генерации контента. Например, сервис может передать текстовое описание и запросить создание изображения, либо передать изображения и запросить создание видео. Настройки, хранящиеся в блоке конфигурации, могут быть использованы для настройки работы блока модели.

2.5 Блок загрузки/выгрузки модели

Блок загрузки/выгрузки модели играет ключевую роль в процессе работы с нейронными сетями, обеспечивая управление весами и состоянием моделей. Он отвечает за эффективную загрузку, сохранение и управление состоянием нейронных сетей как до начала обучения, так и в процессе работы модели, обеспечивая возможность быстрого восстановления обученных состояний. Этот блок критически важен для повышения эффективности работы системы, так как позволяет экономить время и ресурсы, избегая повторного обучения моделей или их перезагрузки после каждого перезапуска системы.

Данный блок напрямую взаимодействует с блоком модели, обеспечивая

загрузку весов перед началом работы модели и сохранение состояния после завершения выполнения задачи. Блок модели инициирует вызовы функций для загрузки или сохранения весов, передавая необходимые параметры, такие как путь к файлам или флаг продолжения обучения.

Блок обучения модели также тесно взаимодействует с блоком загрузки/выгрузки. Во время обучения модель может быть периодически сохраняема, чтобы в случае прерывания обучения можно было восстановить ее состояние.

2.6 Блок обучения модели

Блок обучения модели является центральным компонентом системы, обеспечивающим эффективное обучение нейронных сетей для выполнения задач генерации контента. Этот блок отвечает за настройку и запуск процесса обучения моделей, включая определение оптимизаторов, расчет потерь и обновление весов нейронных сетей. Также он играет ключевую роль в адаптации моделей к различным данным и улучшении их результатов в ходе многократных итераций.

Данный блок должен получать подготовленные данные для обучения, которые могут быть переданы через блок данных для обучения. Обучение модели обычно делится на несколько эпох, где каждая эпоха представляет собой полный цикл обработки всех обучающих данных.

Для упрощения анализа прогресса обучения можно использовать визуализацию, например, через графики, отображающие изменения потерь или точности с течением времени.

Различные параметры обучения, такие как количество эпох, размер батча, скорость обучения, выбираются и передаются в блок обучения модели из блока конфигурации.

Блок обучения использует функции блока загрузки/выгрузки модели для сохранения и загрузки состояния модели до и после обучения, а также для обновления весов на каждом этапе.

2.7 Блок данных для обучения

Блок данных для обучения является компонентом системы, отвечающим за подготовку и управление данными, которые используются для тренировки моделей. Этот блок занимается не только загрузкой данных, но и их предварительной обработкой, хранением и разделением на обучающие, валидационные и тестовые выборки. Все это необходимо для того, чтобы нейронные сети могли эффективно обучаться и правильно обрабатывать входную информацию для генерации изображений и видео.

Блок данных для обучения активно взаимодействует с блоком обучения модели, передавая данные для дальнейшей обработки.

В случае, если обучение модели включает обработку текстовых данных, например, для создания изображений или видео на основе текстовых

описаний, блок данных для обучения должен проводить токенизацию текста, его векторизацию и преобразование в формат, который может быть принят моделью.

Для улучшения качества обработки текста может быть использован предобученный эмбеддинг, который представляет текст в виде плотных векторов.

2.8 Блок пользовательского интерфейса

Блок пользовательского интерфейса (ПИ) представляет собой важную часть системы, обеспечивающую взаимодействие пользователя с программным продуктом. Он выполняет роль «мостика» между пользователем и ядром системы, которое включает в себя нейронные сети для генерации изображений и видео, а также различные сервисы, выполняющие вычислительные операции.

Основная цель блока пользовательского интерфейса – предоставление пользователю интуитивно понятного и удобного способа взаимодействовать с системой, на основе которого он может отправлять запросы, получать результаты и управлять процессом генерации контента.

Основные функции блока пользовательского интерфейса:

- предоставление интерфейса для ввода данных;
- ввод текста для генерации контента;
- загрузка изображения для генерации видео;
- выбор некоторых параметров генерации;
- отправка запросов в сервисы;
- получение и отображение результатов;
- обработка ошибок и сообщений пользователю.

Так как процесс генерации изображений или видео может занимать некоторое время, интерфейс должен поддерживать асинхронные операции и отображать прогресс выполнения задачи. Пользователь должен видеть, что система выполняет задачу, и иметь возможность продолжить работу с интерфейсом.

Блок пользовательского интерфейса напрямую взаимодействует с блоком сервисов. Он отправляет запросы на выполнение определенных действий, таких как генерация изображения или видео, и получает результаты. Сервисы выполняют логику обработки данных, а ПИ лишь представляет результаты пользователю.

Таким образом, данный блок играет важную роль в обеспечении удобного и интуитивно понятного взаимодействия между пользователем и системой.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Функциональное проектирование направлено на создание структуры программы, которая будет эффективно и корректно выполнять все требуемые функции. Этот раздел содержит описание основных модулей, классов и методов, которые будут разработаны для достижения целей проекта, а также подходы, используемые для их реализации. Также рассмотрены потоки данных между компонентами системы, структура и взаимосвязи классов.

После анализа требуемых для реализации программного продукта функций, было решено разбить программу на следующие модули (в скобках указан блок из системного проектирования, на основе которого спроектирован модуль):

- модуль пользовательского интерфейса (блок пользовательского интерфейса);
- модуль сервисов (блок сервисов);
- модуль обработки данных (блок сервисов);
- модуль логирования (блок логирования);
- модуль конфигурации (блок конфигурации);
- модуль загрузки/выгрузки модели (блок загрузки/выгрузки модели);
- модуль дискриминатора (блок обучения модели);
- модуль генератора (блок обучения модели);
- модуль набора данных (блок данных для обучения);
- модуль семплирования (блок обучения модели);
- модуль обучения (блок обучения модели).

Разработка приложения ведется преимущественно с использованием объектно-ориентированной парадигмы (ООП). Как следствие каждый из модулей, который описан в контексте данного дипломного проекта, представляет собой класс или ряд классов, содержащих ряд методов.

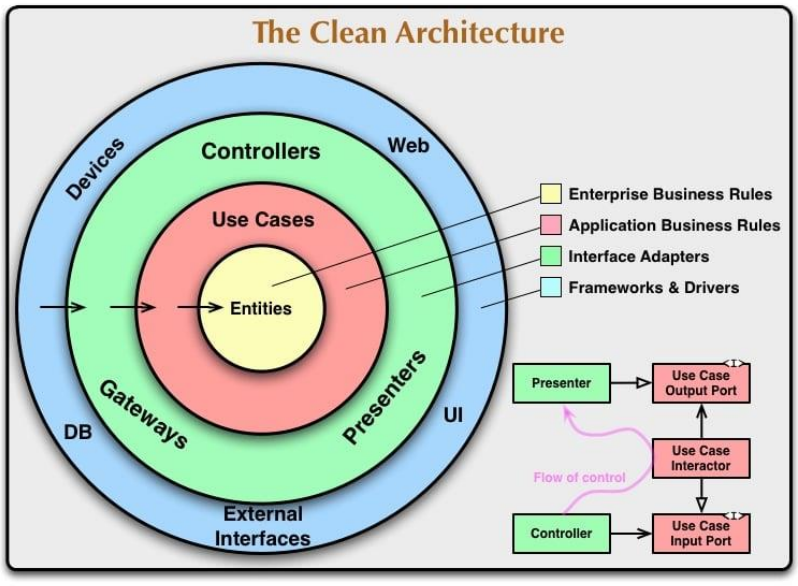

Рисунок 3.1 – Структура «Чистой архитектуры» приложения [23]

При этом в процессе разработки соблюдается стандарт PEP 8, который определяет стиль написания кода на языке Python. Следование этому стандарту способствует улучшению читаемости и поддерживаемости кода, а также облегчает работу в команде, так как все разработчики придерживаются единого стиля.

Кроме того, в проекте применяется подход чистой архитектуры [23], который акцентирует внимание на разделении ответственности и независимости компонентов (рисунок 3.1).

Чистая архитектура позволяет изолировать бизнес-логику от деталей реализации, таких как пользовательские интерфейсы или внешние системы. Это дает возможность легко тестировать и изменять различные части приложения, не затрагивая другие компоненты, что в свою очередь способствует более эффективному процессу разработки и снижению вероятности появления ошибок.

Разработанные классы и их взаимодействие отображены на диаграмме классов ГУИР.400201.042 РР.1, а также на диаграмме последовательности ГУИР.400201.042 РР.2, которая иллюстрирует процесс регистрации, авторизации и создания видео.

3.1 Модуль пользовательского интерфейса

3.1.1 Модуль пользовательского интерфейса (ПИ) представляет собой важный компонент системы, предоставляющий пользователю удобный способ взаимодействия с программой для генерации видео на основе изображений и текстов. Интерфейс был разработан с использованием фреймворка Streamlit, который позволяет быстро и эффективно создавать веб-приложения для работы с нейронными сетями и визуализацией данных.

Основная цель модуля ПИ – предоставить пользователю возможность ввести текстовое описание или загрузить изображение, после чего происходит генерация видео. Весь функционал интерфейса структурирован таким образом, чтобы быть интуитивно понятным, с простыми элементами ввода, визуализацией загруженных данных и возможностью запуска процесса генерации.

В модуле ПИ предусмотрено поле для ввода текста, который пользователь может использовать для генерации видео. Это поле является одним из ключевых способов взаимодействия с системой, так как система генерации видео из текста требует от пользователя описания, на основе которого создается видеоконтент.

Интерфейс также предоставляет возможность загружать изображения. Эти изображения могут использоваться для генерации видео как отдельный источник данных. Загруженное изображение затем визуализируется в интерфейсе, что дает пользователю возможность убедиться в правильности выбранного файла перед запуском процесса генерации.

Основной триггер для начала генерации – это кнопка «Generate». После нажатия этой кнопки сервисы, подключенные к интерфейсу, начинают

процесс генерации видео на основе введенного текста или загруженного изображения. Это взаимодействие с серверной частью происходит асинхронно, что позволяет интерфейсу оставаться отзывчивым во время работы системы.

Размещение кнопки в центральной колонке интерфейса помогает улучшить визуальную структуру и сделать интерфейс более удобным для пользователя. Для удобства пользователей и лучшего понимания цели интерфейса используются стилизованные заголовки и описания.

На рисунке 3.2 представлен макет данного приложения.

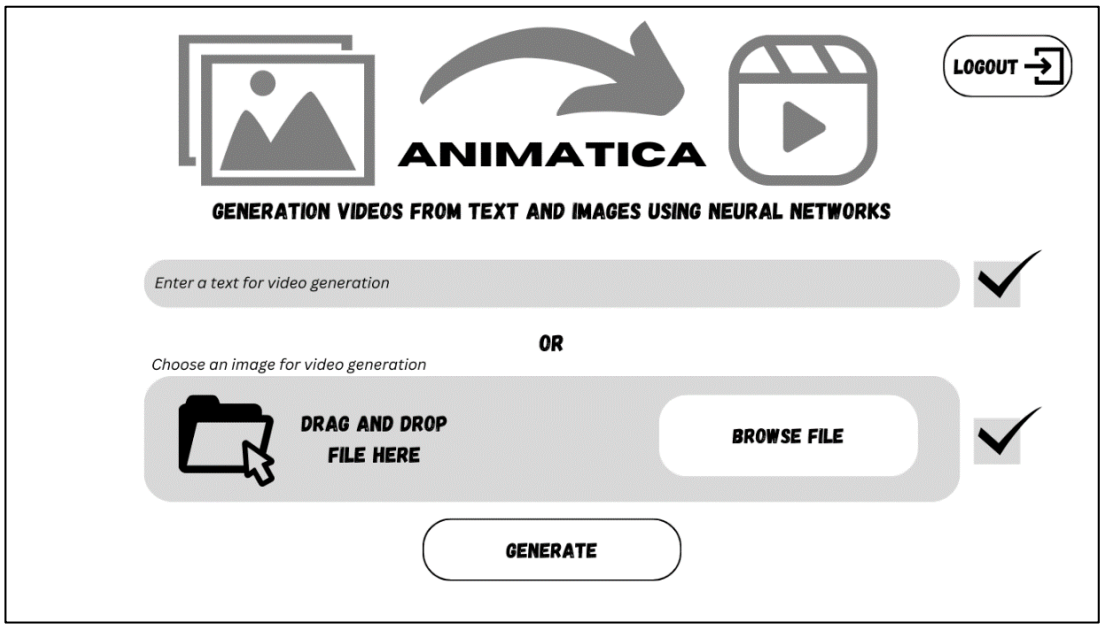

Рисунок 3.2 – Макет приложения для генерации видео после входа в аккаунт

3.1.2 Класс `UserInterface` является основным контейнером для всех функций и компонентов пользовательского интерфейса. В этом классе определены основные методы для отображения различных элементов интерфейса, таких как ввод текста, загрузка изображений и кнопки:

- `__init__()` – конструктор класса, где инициализируются основные параметры приложения, такие как заголовок страницы и описание;
- `display_header()` – метод для отображения заголовка и краткого описания приложения;
- `get_text_input()` – метод, который позволяет пользователю ввести текст для генерации видео;
- `load_image()` – метод, обрабатывающий загрузку изображения пользователем через Streamlit;
- `display_generate_button()` – метод, отображающий кнопку для начала процесса генерации видео.

3.1.3 Класс `VideoGenerationService` отвечает за интеграцию с сервисами, которые занимаются генерацией видео. Он вызывает функции

генерации на основе данных, полученных от пользователя (текста или изображения), и предоставлять результат.

Описание класса:

- `__init__()` – инициализируется модель генерации видео, которая используется для создания видео на основе текста или изображений;

- `generate_video_from_text()` – генерирует видео на основе текстового ввода;

- `generate_video_from_image()` – генерирует видео на основе загруженного изображения.

3.2 Модуль сервисов

Модуль сервисов является ключевой частью архитектуры системы, основанной на паттерне API, которая реализуется с помощью FastAPI. Он предназначен для обработки запросов, взаимодействия с базой данных и выполнения бизнес-логики. Для обеспечения высокой структурированности и читаемости кода используются абстракции в виде интерфейсов и классов для реализации ключевых функциональных элементов системы, таких как варианты использования (юзкейсы), репозитории и сущности. Ниже приводится описание предположительной структуры кода с использованием предложенных классов и интерфейсов.

3.2.1 Интерфейс `IUseCase` представляет собой абстракцию для реализации бизнес-логики. Каждый конкретный юзкейс, например, генерация видео или изображения, реализовывает данный интерфейс.

Описание методов:

- `execute()` – абстрактный метод, который должен быть реализован в каждом конкретном юзкейсе, принимающий объект запроса и возвращающий результат обработки.

3.2.2 Интерфейс `IDBRepository` определяет методы для взаимодействия с базой данных. В него могут входить стандартные CRUD-операции (англ. Create-Read-Update-Delete), такие как сохранение и получение данных:

- `save()` – метод для сохранения сущности в базе данных;

- `get_by_id()` – метод для получения сущности из базы данных по её идентификатору.

3.2.3 Класс `UserEntity` представляет собой доменную сущность пользователя, которая используется в бизнес-логике. Она содержит информацию о пользователе, используется в бизнес-логике.

Класс `UserOrm` – это модель, используемая для взаимодействия с базой данных. Он представляет ORM-модель пользователя для взаимодействия с базой данных.

3.2.4 Классы `Request` и `Response` являются объектами для передачи данных между слоями системы (DTO – англ. Data Transfer Objects) и присутствуют в каждом отдельном юзкейсе.

Класс `Request` представляет запрос от клиента или от других модулей системы. Включает данные, необходимые для обработки запроса.

Класс `Response` представляет ответ, содержащий информацию об успешности операции, сообщение и данные результата.

3.2.5 Каждый юзкейс реализует интерфейс `IUseCase` и отвечает за выполнение конкретной задачи, связанной с генерацией контента (видео или изображений).

Класс `GenerateVideoByImageUseCase` отвечает за генерацию видео на основе изображения. Включает следующий метод:

- `execute()` – получает изображение из запроса, генерирует видео с использованием модели, сохраняет результат в базу данных через репозиторий и возвращает ответ.

Класс `GenerateVideoByTextUseCase` отвечает за генерацию видео на основе текста. Включает следующий метод:

- `execute()` – обрабатывает текстовый запрос, генерирует видео с помощью модели и сохраняет результат.

Кроме генерации видео по изображению и тексту, возможны и другие юзкейсы:

- `GenerateRandomVideoUseCase` – для генерации случайного видео;
- `GenerateImageByTextUseCase` – для генерации изображения на основе текста;
- `GenerateRandomImageUseCase` – для генерации случайного изображения.

3.3 Модуль обработки данных

Модуль обработки данных является важной частью системы, поскольку он отвечает за подготовку, преобразование и передачу данных между различными компонентами. В проекте модуль обработки данных реализован как на стороне FastAPI (серверная часть), так и в части процесса обучения нейронных сетей. Каждая из этих частей выполняет свои задачи по работе с данными, обеспечивая их корректное использование для генерации изображений и видео, обучения модели и взаимодействия с пользователем.

3.3.1 На стороне FastAPI модуль обработки данных (класс `DataProcessor`) отвечает за работу с входными и выходными данными API-запросов, которые поступают от клиента или пользователя. Входящие данные могут быть изображениями, текстом или конфигурациями для генерации контента, которые должны быть подготовлены и переданы в соответствующие

модули для дальнейшей обработки.

Методы класса:

- `process_image()` – преобразует изображение, переданное в запросе, в нужный формат для дальнейшей обработки моделью (например, преобразование в тензор или другой формат);
- `process_text()` – преобразует текстовый запрос в формат, который может быть использован для генерации контента.

3.3.2 В процессе обучения класс `TrainingDataProcessor` отвечает за подготовку тренировочных данных, их аугментацию и передачу в нейронную сеть. Этот класс важен для оптимизации процесса обучения и повышения качества модели.

Описание методов:

- `process_training_image()` – метод для предобработки изображений перед подачей в нейронную сеть (использует стандартные преобразования, такие как изменение размера, нормализация и преобразование в тензор).

Таким образом, модуль обработки данных играет связующую роль в проекте, обеспечивая корректное преобразование данных между интерфейсом, серверной частью и моделью, что позволяет системе функционировать эффективно.

3.4 Модуль логирования

Модуль логирования обеспечивает возможность отслеживания процесса работы как сервера FastAPI, так и обучения нейронной сети. Логирование необходимо для мониторинга ошибок, анализа производительности, отладки и сохранения информации о ключевых этапах выполнения системы.

Модуль логирования состоит из двух основных классов:

- класс для логирования на стороне FastAPI;
- класс для логирования на стороне обучения модели.

3.4.1 Класс `APILogger` отвечает за ведение логов запросов и ответов, возникающих в процессе работы сервера. Это позволяет отслеживать обращения пользователей к API, фиксировать ошибки, предупреждения, а также время выполнения операций. Логирование на сервере особенно важно для анализа производительности системы и отладки.

Описание класса:

- `log_request()` – записывает информацию о входящих HTTP-запросах, включая метод и URL, что позволяет отслеживать использование API;
- `log_response()` – записывает информацию об ответах сервера, включая статус-код, что важно для мониторинга успешности или неуспешности выполнения запросов;

- log_error() – записывает ошибки, возникшие на сервере, что облегчает отладку и выявление проблем;
- log_info() – записывает произвольные информационные сообщения для общего мониторинга работы сервера.

3.4.2 В процессе обучения нейронной сети логирование играет важную роль для отслеживания метрик и этапов обучения. Класс ModelLogger фиксирует прогресс обучения, значения потерь, точности и другие ключевые метрики, что позволяет анализировать эффективность обучения и корректировать гиперпараметры при необходимости.

Описание класса:

- log_epoch() – записывает метрики, такие как потери (loss) и точность (accuracy), по завершении каждой эпохи, что позволяет отслеживать прогресс обучения;
- log_batch() – записывает потери по каждому батчу в процессе обучения, что помогает детализировать информацию о поведении модели внутри эпох;
- log_hyperparameters() – записывает используемые гиперпараметры (например, скорость обучения и размер батча), что важно для последующего анализа влияния этих параметров на результат;
- log_training_completion() – записывает успешное завершение процесса обучения модели;
- log_error() – записывает ошибки, возникшие в процессе обучения, что важно для отладки.

3.5 Модуль конфигурации

Программа оперирует большим количеством параметров и настроек, которые задаются через константы. Далее перечислены некоторые константы:

- IMAGE_DATASET_PATH – путь к набору данных для обучения;
- IMAGE_BATCH_SIZE – размер пакета изображений для обучения;
- VIDEO_BATCH_SIZE – размер пакета видеороликов для обучения.

Параметры изображений и видео:

- IMAGE_SIZE – размер изображения после изменения;
- VIDEO_LENGTH – длина видео;
- N_CHANNELS – количество каналов входных данных;
- EVERY_NTH – выбор каждого n-го кадра для обучения видеоданных.

Параметры моделей дискриминаторов:

- IMAGE_DISCRIMINATOR_TYPE – тип дискриминатора для изображений;
- VIDEO_DISCRIMINATOR_TYPE – тип дискриминатора для видео.

Опции использования и настройки:

- USE_INFOGAN – использование InfoGAN потерь;

- USE_CATEGORIES – использование категорий для обучения;

- USE_NOISE – добавление шумов в данные;

- NOISE_SIGMA – параметр, задающий амплитуду шума (при использовании шумов).

Генеративные параметры:

- DIM_Z_CONTENT – размерность латентного пространства для контента;

- DIM_Z_MOTION – размерность латентного пространства для движения;

- DIM_Z_CATEGORY – размерность категорий.

Настройки для обучения и логирования:

- NUM_EPOCHS – количество эпох для обучения модели;

- PRINT_EVERY – вывод информации каждые n итераций;

- BATCH_COUNT – количество пакетов данных для обучения.

Для оценки эффективности модели используются переменные для подсчета потерь. В ходе обучения генератора и дискриминатора вычисляются следующие параметры:

- потери дискриминатора на реальных данных (lossD_real);

- потери дискриминатора на сгенерированных данных (lossD_fake);

- общие потери дискриминатора (lossD_total = lossD_real + lossD_fake);

- потери генератора (lossG).

Эти переменные позволяют отслеживать, насколько успешно генератор обучается на основе ответов дискриминатора, что важно для корректной работы модели.

Класс Config отвечает за загрузку и управление конфигурацией приложения. Также он содержит параметры конфигурации для подключения к базе данных.

3.6 Модуль загрузки/выгрузки модели

Модуль загрузки/выгрузки модели играет важную роль в управлении сохранением и восстановлением весов нейронной сети в процессе работы системы. Он позволяет загружать уже обученные веса для использования в генерации контента, а также сохранять обновленные веса в процессе обучения. Этот модуль обеспечивает устойчивость и переносимость модели, позволяя сохранять текущее состояние и восстанавливать его при необходимости.

Данный модуль реализован в виде одного класса – ModelManager, который объединяет функционал для загрузки готовых весов модели, их сохранения и обновления в ходе тренировки. Модуль тесно взаимодействует с другими блоками системы, такими как обучение и сервисы, для обеспечения корректного выполнения генерации и обучения.

Описание функциональности класса:

- __init__() – инициализация класса, принимает на вход

инициализированную архитектуру модели (без весов) и путь к файлу с сохраненными весами, если такой файл существует;

- `load_weights()` – метод, который загружает предобученные веса в модель из указанного файла;

- `save_weights()` – метод, сохраняющий текущие веса модели в указанный файл в формате, который поддерживается библиотекой PyTorch;

- `get_model()` – метод возвращает текущую модель с загруженными весами, которую могут использовать другие сервисы для выполнения задач генерации или обучения.

3.7 Модуль генератора

Модуль генератора отвечает за создание видео и изображений на основе случайных латентных переменных, описывающих движение, содержание и категорию объектов. Этот модуль реализован с использованием библиотеки PyTorch и включает в себя класс `VideoGenerator`, который наследует функциональность нейронной сети через класс `nn.Module`. Генератор является ключевым компонентом системы, обеспечивающим создание контента на основе заранее обученной модели.

Класс `VideoGenerator` – это нейронная сеть, состоящая из рекуррентного слоя для работы с латентными переменными движения и нескольких транспонированных сверточных слоев для генерации видео из латентного пространства. Модель поддерживает генерацию как видео, так и статических изображений, основанных на различных латентных факторах (содержание, категория и движение).

Основные параметры:

- `n_channels` – количество каналов на выходе (например, 3 для цветного видео или изображения RGB);

- `dim_z_content` – размерность латентного пространства для описания содержания (например, характеристики объектов на видео);

- `dim_z_category` – размерность латентного пространства для категорий (например, классов объектов);

- `dim_z_motion` – размерность латентного пространства для движения объектов;

- `video_length` – длина видео в кадрах;

- `ngf` – количество фильтров, используемых в слоях генератора (по умолчанию 64).

Далее описываются основные компоненты генератора.

Для генерации латентных переменных движения используется рекуррентная сеть (GRU), которая обрабатывает случайные шумовые векторы на каждом шаге временной последовательности. Это позволяет сохранять последовательность и контекст движения.

Структура сети включает серию транспонированных сверточных слоев, которые постепенно увеличивают размер латентных векторов до размера

видео. Эти слои используют активацию ReLU и нормализацию батча (метод BatchNorm), что улучшает стабильность и качество сгенерированных видео.

Описание методов класса:

- `__init__()` – инициализация класса с определением всех необходимых параметров для генерации видео;
- `sample_z_m()` – генерация латентных переменных для движения объектов на видео с использованием GRU;
- `sample_z_categ()` – генерация латентных переменных для категорий объектов (например, классов), где для каждого видео создается one-hot вектор, который повторяется на всех кадрах;
- `sample_z_content()` – генерация латентных переменных для содержания объектов на видео, где для каждого кадра содержание повторяется через видео;
- `sample_z_video()` – комбинирует латентные переменные для движения, содержания и категорий, создавая полный набор латентных переменных для генерации видео;
- `sample_videos()` – генерация видеороликов с заданным количеством выборок и длиной видео, возвращает видео и метки категорий;
- `sample_images()` – генерация статических изображений, которые выбираются случайным образом из латентных переменных;
- `get_gru_initial_state()` – инициализирует начальное состояние GRU для движения;
- `get_iteration_noise()` – генерация случайного шума для каждой итерации GRU, что задает изменение движения между кадрами.

3.8 Модуль дискриминатора

Модуль дискриминатора предназначен для обучения модели различать реальные и сгенерированные данные. В контексте генеративно-состязательных сетей, дискриминатор играет ключевую роль, оценивая вероятность того, что входные данные (изображения или видео) являются реальными или сгенерированными.

3.8.1 `ImageDiscriminator` – дискриминатор для изображений. Этот класс построен на сверточной архитектуре для анализа двумерных изображений. Основные блоки включают сверточные слои с фильтрами разной размерности и функцию активации LeakyReLU, которая лучше справляется с отрицательными значениями по сравнению с обычной ReLU.

Используется опциональное добавление шума в слои через класс `Noise`, что помогает избежать переобучения и делает модель более устойчивой.

Модель уменьшает размерность изображения на каждом слое, что помогает дискриминатору находить сложные паттерны и различия в данных.

Данный класс имеет следующие переменные:

- use_noise – переменная, которая указывает, нужно ли добавлять шум в слои с помощью класса Noise;

- main – последовательность слоев, включая слои для добавления шума, сверточные слои, нормализацию и активационные функции.

Методы класса:

- __init__() – конструктор класса, который инициализирует все необходимые слои и переменные;

- forward() – Пропускает вход через основной блок слоев, который включает все сверточные слои, шум и активации.

3.8.2 PatchImageDiscriminator – дискриминатор для изображений, работающий по «патчам» (пакетам). Этот класс отличается от стандартного дискриминатора тем, что разделяет изображения на небольшие патчи и анализирует их. Модель пытается идентифицировать реальность не целого изображения, а небольших его участков (патчей), что может повысить точность работы.

Это улучшает способность дискриминатора оценивать мелкие детали, такие как текстуры и малые объекты.

Класс имеет такие же переменные и методы, как у ImageDiscriminator, но со своими значениями и архитектурой для последовательности, что хранится в переменной main.

3.8.3 PatchVideoDiscriminator – дискриминатор для видео, работающий по патчам. Подобно PatchImageDiscriminator, этот дискриминатор применяет архитектуру на основе сверточных 3D-слоев, что позволяет ему обрабатывать видеоданные (последовательности кадров).

Используются сверточные слои с тремя измерениями (ширина, высота и время), чтобы захватывать как пространственные, так и временные зависимости.

Как и в случае с изображениями, этот дискриминатор анализирует патчи видео, что помогает ему оценивать реалистичность мелких сегментов видеопоследовательностей.

Класс повторяет переменные и методы класса ImageDiscriminator, но со своими значениями и архитектурой для перменной main. Также класс имеет дополнительные переменные:

- n_channels – количество каналов на входе;

- n_output_neurons – количество выходные нейронов;

- bn_use_gamma – переменная для использования/неиспользования гамма-значений в нормализации батча.

3.8.4 VideoDiscriminator – основной дискриминатор для работы с видео. В отличие от PatchVideoDiscriminator, этот класс работает с видео на уровне всей последовательности, а не отдельных патчей.

Он включает в себя сверточные слои с 3D-фильтрами, которые

обрабатывают как пространственные, так и временные зависимости.

Для стабилизации процесса обучения используется нормализация по батчам (метод `BatchNorm3d`). Дополнительно используется шум для увеличения устойчивости модели.

Класс имеет такие же методы и переменные по назначению, как и у класса `ImageDiscriminator`.

3.8.5 `CategoricalVideoDiscriminator` – категориальный дискриминатор для видео. Наследуется от `VideoDiscriminator`, но добавляет категориальные переменные, которые позволяют дискриминатору учитывать принадлежность к различным классам данных (например, различные типы объектов в видео).

Метод `split` разделяет выходные данные на два компонента: метки для задачи дискриминации и категориальные переменные, которые могут быть использованы для дальнейшего анализа.

Кроме этого, класс имеет переменную `dim_categorical` для хранения размерности категориальных выходных данных, добавленных в выходное значение (например, для обработки меток или классов).

3.9 Модуль набора данных

Данный модуль включает три ключевых класса: `VideoFolderDataset`, `ImageDataset` и `VideoDataset`. Он предназначен для работы с наборами данных, которые содержат как изображения, так и видео. Основные функции включают загрузку, обработку видео и изображений, а также выборку кадров или последовательностей видео для дальнейшего использования в задачах машинного обучения и глубокого обучения.

3.9.1 Класс `VideoFolderDataset` предназначен для загрузки видео и изображений из указанной папки и, при необходимости, для кэширования данных. Класс автоматически вычисляет длину каждого видео (в кадрах) и сохраняет информацию о всех видеофайлах в папке.

Основные переменные:

- `folder` – путь к папке с изображениями/видео;
- `cache` – путь для кэширования данных, что позволяет избежать повторной обработки данных при последующих запусках;
- `min_len` – минимальная длина видео в кадрах, необходимая для включения в набор данных;
- `images` – список кортежей с путями к изображениям и метками;
- `lengths` – список длин каждого видео в кадрах;
- `cumsum` – накопленная сумма длин видео для быстрой выборки кадра по индексу.

Основные методы:

- `__init__()` – инициализация класса (загружает набор данных из

указанной папки, подсчитывает количество кадров в каждом видео и сохраняет их в кэш);

- `__getitem__()` – возвращает изображение (кадр) и его метку на основе индекса;

- `__len__()` – возвращает количество доступных видео в наборе данных.

3.9.2 Класс `ImageDataset` является оберткой над `VideoFolderDataset` и позволяет выбирать отдельные кадры из видео, при этом поддерживая возможность применения различных трансформаций к кадрам.

Основные переменные:

- `dataset` – набор данных, переданный в качестве аргумента, обычно это объект `VideoFolderDataset`;

- `transforms` – трансформации, применяемые к кадрам изображений.

Основные методы:

- `__getitem__()` – возвращает отдельный кадр из видео на основе индекса;

- `__len__()` – возвращает общее количество кадров во всех видео в наборе данных.

3.9.3 Класс `VideoDataset` позволяет выбирать последовательности кадров из видео фиксированной длины. Поддерживается выборка кадров через заданное количество (шаг), что позволяет формировать подмножества кадров для задач, связанных с обучением на видеоданных (например, видео-классификация).

Основные переменные:

- `dataset` – исходный набор данных;

- `video_length` – количество кадров в последовательности, которую нужно извлечь;

- `every_nth` – шаг выборки кадров;

- `transforms` – трансформации, применяемые к извлеченной последовательности видео.

Основные методы:

- `__getitem__()` – возвращает последовательность кадров из видео;

- `__len__()` – возвращает количество видео в наборе данных.

3.10 Модуль семплирования

Данный модуль предоставляет два класса – `ImageSampler` и `VideoSampler` – которые используются для семплирования изображений и видео соответственно. Эти классы предназначены для извлечения данных из набора данных с возможностью применения различных трансформаций, а

также для случайного выбора последовательностей кадров из видео.

3.10.1 Класс `ImageSampler` отвечает за выборку отдельных изображений из набора данных и их обработку. Он поддерживает применение трансформаций к изображениям перед их возвратом.

Основные переменные:

- `dataset` – набор данных, содержащий изображения;
- `transforms` – трансформации, которые будут применяться к изображениям.

Основные методы:

- __init__() – инициализация класса, установка набора данных и опционально переданных трансформаций.
- __getitem__() – извлекает данные по индексу;
- __len__() – возвращает количество элементов в наборе данных, основываясь на размере данных, извлекаемых по первому ключу.

3.10.2 Класс `VideoSampler` предназначен для семплирования последовательностей кадров из видео. Он поддерживает выборку кадров через заданные промежутки и случайное определение начальной точки выборки.

Основные переменные:

- `dataset` – набор данных, содержащий информацию о видео;
- `video_length` – длина последовательности кадров, которую необходимо извлечь;
- `every_nth` – шаг выборки кадров;
- `transforms` – трансформации, применяемые к извлеченным последовательностям кадров.

Основные методы:

- __init__() – инициализация класса с установкой набора данных, длины видео, шага выборки и трансформаций;
- __getitem__() – извлекает последовательность кадров из видео;
- __len__() – возвращает количество уникальных видео в наборе данных.

3.11 Модуль обучения

Этот модуль содержит классы и функции, необходимые для обучения GAN, работающей с изображениями и видео. Класс `Trainer` используется для тренировки генератора и дискриминаторов, поддерживает логи, обработку батчей данных и подсчет потерь.

Модуль имеет отдельные функции преобразования, не связанные с классом `Trainer`:

- `images_to_numpy()` – преобразует тензор изображений в формат `NumPy`, изменяя значения пикселей от диапазона [-1, 1] к диапазону [0, 255];
- `videos_to_numpy()` – аналогичная функция для преобразования видеотензоров в формат `NumPy`;

- one_hot_to_class() – преобразует one-hot представление меток в классы (целые числа).

Класс Trainer отвечает за тренировку генератора и дискриминаторов. Он реализует как тренировку дискриминаторов, так и генератора на основе ошибок GAN и категориальной кросс-энтропии (если используется InfoGAN).

Основные параметры:

- image_sampler – объект, предоставляющий батчи изображений для тренировки;
- video_sampler – объект, предоставляющий батчи видео;
- log_interval – частота вывода логов по итерациям;
- train_batches – количество батчей для каждой эпохи;
- log_folder – папка для сохранения логов и моделей;
- use_cuda – флаг для использования GPU (CUDA);
- use_infogan – флаг для использования архитектуры InfoGAN, где генератор учится генерировать распознаваемые категории;
- use_categories – флаг для включения категориальной классификации дискриминатором видео.

Основные методы:

- ones_like() и zeros_like() – создают тензоры заполненные единицами или нулями, используемые для меток в задачах бинарной классификации (реальный или поддельный пример);
- compute_gan_loss() – вычисляет потери для генератора и дискриминатора на основе сгенерированных и реальных данных;
- sample_real_image_batch() и sample_real_video_batch() – выбирают батчи изображений и видео из семплеров;
- train_discriminator() – обучает дискриминатор на реальных и поддельных данных;
- train_generator() – обучает генератор на основе обратной связи от дискриминаторов;
- train() – обучает GAN.

Оптимизаторы (Adam) создаются для генератора и дискриминаторов с фиксированной скоростью обучения и коэффициентом β. Логи (потери генератора и дискриминаторов) сохраняются и выводятся через заданные интервалы. Также сохраняются промежуточные версии генератора каждые 5 эпох.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ НА РЫНКЕ ПРОГРАММНОГО СРЕДСТВА ДЛЯ АНИМАЦИИ ИЗОБРАЖЕНИЙ НА ОСНОВЕ НЕЙРОННЫХ СЕТЕЙ

7.1 Характеристика программного средства, разрабатываемого для реализации на рынке

В современных условиях креативных индустрий растет потребность в инструментах, позволяющих быстро и качественно преобразовывать текстовую информацию и статичные изображения в динамические видеоматериалы. Разрабатываемое программное средство направлено на решение этой задачи с помощью нейронных сетей, обеспечивая возможность создания анимации на основе текстовых описаний и изображений, что значительно упрощает процессы видео- и контент-продакшна.

Данный проект позволит автоматизировать процесс генерации видео, тем самым сократив временные затраты на создание анимации. Предполагается улучшение качества и реалистичности генерируемых видеоматериалов, а также поддержка различных форматов изображений и видео для гибкости работы.

Возможные области применения:

- реклама и маркетинг (создание рекламных роликов, видео для социальных сетей);
- кинопроизводство и анимация (быстрая генерация анимаций и видеороликов);
- разработка игр (генерация анимаций для игровых персонажей и объектов);
- визуализация концептов в дизайне и архитектуре.

В функции программного средства входит:

- генерация изображений на основе текстовых описаний;
- преобразование изображений в видеопоследовательности;
- постобработка видеоматериалов для улучшения их качества (например, сглаживание, повышение разрешения);
- экспорт полученных видеороликов в различных форматах для дальнейшего использования;
- взаимодействие с пользователями через удобный интерфейс для настройки и управления процессами генерации.

Программное средство будет доступно пользователям через веб-приложение с гибкой моделью подписки, что позволит выбрать наиболее подходящий тарифный план в зависимости от частоты и объема использования. Благодаря этому подходу, приложение будет доступно широкой аудитории, а разнообразие подписок, от базовой до премиума, обеспечит гибкость и удовлетворение потребностей различных категорий клиентов, от фрилансеров до крупных компаний.

7.2 Расчет инвестиций в разработку программного средства для его реализации на рынке

7.2.1 Расходы на основную заработную плату рассчитываются на основе состава и численности команды, месячного оклада каждого сотрудника, объема выполненных ими задач, а также с учетом размера премиальных. Такой расчет может быть выполнен с использованием следующей формулы:

$$З_о = К_{пр} \cdot \sum_{i=1}^{n} З_{чi} \cdot t_i, \quad (7.1)$$

где $К_{пр}$ – коэффициент премий и иных стимулирующих выплат;

n – категории исполнителей, занятых разработкой;

$З_{чi}$ – часовой оклад плата исполнителя i-й категории, р.;

t_i – трудоемкость работ, выполняемых исполнителем i-й категории, ч.

Часовая ставка каждого сотрудника рассчитывается путем деления его месячного оклада на количество рабочих часов в месяц. В расчетах принимается, что месячное количество рабочих часов составляет 160. Расчет затрат на оплату труда разработчиков приведён в таблице 7.1.

Таблица 7.1 – Расчет затрат на основную заработную плату разработчиков

Категория исполнителя	Месячный оклад, р.	Часовой оклад, р.	Трудоемкость работ, ч.	Итого, р.
Инженер по машинному обучению	3800	23,75	160	3800
Инженер-программист	3500	21,88	160	3500
Тестировщик	2200	13,75	96	1320
Проектный менеджер	2150	13,44	40	537,60
Итого				9158
Премия и иные стимулирующие выплаты, 20%				1831,60
Всего затрат на основную заработную плату разработчиков				10989,60

В команде проекта ключевую роль играет инженер по машинному обучению, который занимается разработкой и настройкой нейросетей. Его задача заключается в создании и обучении генеративных моделей, отвечающих за преобразование изображений и текста в видео. Специалист по машинному обучению настраивает архитектуру моделей, подбирает гиперпараметры и оптимизирует их работу, обеспечивая высокую производительность и качество генерируемого контента.

Важным участником процесса является инженер-программист, который сосредоточен на разработке интерфейса и серверной части приложения. Он интегрирует обученные модели в веб-платформу, разрабатывает пользовательские интерфейсы и обеспечивает стабильное функционирование

системы. Этот специалист также отвечает за настройку инфраструктуры для поддержания приложения в онлайн-режиме, обеспечивая его масштабируемость и доступность.

Для обеспечения качества и надёжности проекта привлекается тестировщик, который проверяет работоспособность всех компонентов системы. Он проводит функциональные тесты на корректность выполнения всех операций, выявляет и устраняет ошибки, а также проверяет приложение на устойчивость к нагрузкам, чтобы гарантировать стабильную работу при высоких пользовательских запросах.

Организацию и координацию работы команды осуществляет проектный менеджер, который управляет ресурсами и следит за соблюдением сроков. Его задача также заключается в контроле качества выполнения задач, управлении рисками и взаимодействии с клиентами. Менеджер собирает обратную связь, вносит коррективы в проект по мере необходимости и поддерживает бесперебойное сотрудничество между членами команды.

Данные о заработных платах перечисленных специалистов были взяты из источников [24 – 27].

7.2.2 Для расчета расходов на дополнительную заработную плату разработчиков используем следующую формулу:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (7.2)$$

где $Н_д$ – норматив дополнительной заработной платы.

Норматив дополнительной заработной платы равен 15%.

7.2.3 Размер отчислений на социальное обеспечение рассчитывается на основе ставки отчислений, которая, согласно действующему законодательству на март 2025 года, составляет 35%.

Определим величину отчислений с помощью следующей формулы:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100}, \quad (7.3)$$

где $Н_{соц}$ – норматив отчислений в ФСЗН.

7.2.4 Прочие расходы вычисляются с учетом норматива, который для данного расчета принимается равным 30%.

С учетом этого рассчитаем прочие расходы по следующей формуле:

$$Р_{пр} = \frac{З_о \cdot Н_{пр}}{100}, \quad (7.4)$$

где $Н_{пр}$ – норматив прочих расходов.

7.2.5 Для расчета расходов на реализацию требуется учитывать норматив данных затрат, который в данном случае принимается равным 3%. Расчет этих расходов производится по следующей формуле:

$$Р_р = \frac{З_о \cdot Н_р}{100}, \quad (7.5)$$

где $Н_р$ – норматив расходов на реализацию.

7.2.6 Общую сумму затрат можно определить как сумму ранее рассчитанных расходов, включая основную заработную плату, дополнительную заработную плату, отчисления на социальные нужды и прочие расходы.

Формула для вычисления общей суммы затрат выглядит следующим образом:

$$З_р = З_о + З_д + Р_{соц} + Р_{пр} + Р_р. \quad (7.6)$$

Итоговые затраты на разработку программного средства рассчитываются с использованием вышеуказанной формулы, и результаты представлены в таблице 7.2.

Таблица 7.2 – Расчет инвестиций на разработку программного средства

Наименование статьи затрат	Формула/таблица для расчета	Значение, р.
1 Основная заработная плата разработчиков	Таблица 7.1	10989,60
2 Дополнительная заработная плата разработчиков	$З_д = \frac{10989,60 \cdot 15}{100}$	1648,44
3 Отчисления на социальные нужды	$Р_{соц} = \frac{(10989,60 + 1638,44) \cdot 35}{100}$	4423,31
4 Прочие расходы	$Р_{пр} = \frac{10989,60 \cdot 30}{100}$	3296,88
5 Расходы на реализацию	$Р_р = \frac{10989,60 \cdot 3}{100}$	329,69
6 Общая сумма затрат на разработку и реализацию	$З_р = 10989,60 + 1648,44 + 4423,31 + + 3296,88 + 329,69$	20687,92

7.3 Расчет экономического эффекта от реализации программного средства на рынке

Оценка экономического эффекта от реализации программного средства позволит определить его коммерческую привлекательность и потенциальную прибыльность. Эта величина зависит от объема продаж, стоимости продукта и затрат на его разработку.

Для определения цены программного средства следует провести анализ аналогичных продуктов на рынке. Аналогичные решения, подобные данному проекту, часто работают по модели подписки. В таблице 7.3 приведены примеры аналогичных продуктов [28 – 39] и их стоимость за использование.

Таблица 7.3 – Цена использования продуктов-аналогов

Название продукта	План подписки	Цена, $/месяц
Invideo	Бизнес	15
Renderforest	Профессиональный	21,10
Wave.video	Создатель	24
Pictory.ai	Профессиональный	39
Synthesys	Создатель	41
AI Studios	Персональный	24
Vidnoz	Бизнес	56,99
Pipio.ai	Профессиональный	16
Colossyan	Бизнес	70
VidAU	Бизнес	45
Fliki	Стандартный	21
Elai	Базовый	23

Из представленной таблицы можно вычислить, что средняя стоимость подписки составляет 33 доллара США, что в пересчете на белорусские рубли составляет 108,05 рублей.

Для расчета прироста чистой прибыли необходимо учесть налог на добавленную стоимость (НДС), который рассчитывается по следующей формуле:

$$НДС = \frac{Ц_{отп} \cdot N \cdot Н_{д.с}}{100 + Н_{д.с}}, \quad (7.7)$$

где N – количество оплат за подписку на программный продукт за год, шт.;

$Ц_{отп}$ – отпускная цена подписки программного средства, р.;

$Н_{д.с}$ – ставка налога на добавленную стоимость, %.

Отпускная цена подписки на программное средство установлена на уровне 80 рублей. Ожидается, что за год количество покупок подписки составит примерно пять тысяч. Ставка налога на добавленную стоимость, действующая на март 2025 года согласно законодательству Республики

Беларусь, составляет 20%. Используя эту информацию, можно рассчитать НДС:

$$НДС = \frac{80 \cdot 5000 \cdot 20}{100 + 20} = 66666{,}67 \text{ р.}$$

После вычисления налога на добавленную стоимость, можно рассчитать прирост чистой прибыли, который разработчик получит от реализации программного продукта. Для этого применяется следующая формула:

$$\Delta П_ч^р = (Ц_{отп} \cdot N - НДС) \cdot Р_{пр} \cdot (1 - \frac{Н_п}{100}), \quad (7.8)$$

где $Ц_{отп}$ – отпускная цена подписки программного средства, р.;

N – количество оплат за подписку на программный продукта за год, шт.;

НДС – сумма налога на добавленную стоимость, р.;

$Р_{пр}$ – рентабильность продаж;

$Н_п$ – ставка налога на прибыль.

Ставка налога на прибыль, в соответствии с действующим законодательством на март 2025 года, составляет 20%. Рентабельность продаж программного продукта установлена на уровне 30%.

Прирост чистой прибыли вычисляется по следующей формуле:

$$\Delta П_ч^р = (80 \cdot 5000 - 180083{,}33) \cdot 30\% \cdot (1 - \frac{20}{100}) = 79999{,}99 \text{ р.}$$

7.4 Расчет показателей экономической эффективности разработки и реализации программного средства на рынке

Для оценки экономической эффективности разработки и внедрения программного продукта на рынок, необходимо провести сравнительный анализ затрат на его разработку и ожидаемого прироста чистой прибыли за год. Если сумма затрат на разработку меньше годового экономического эффекта, то можно заключить, что инвестиции окупятся менее чем за один год.

Оценка эффективности инвестиций проводится с использованием расчета рентабельности инвестиций (Return on Investment, ROI). Формула для вычисления ROI следующая:

$$ROI = \frac{\Delta П_ч^р - З_р}{З_р}, \quad (7.9)$$

где $\Delta П_ч^р$ – прирост чистой прибыли, полученной от реализации программного средства на рынке, р;

$З_р$ – затраты на разработку и реализацию программного средства, р.

$$ROI = \frac{216100 - 20687,92}{20687,92} \cdot 100\% = 286,69\%$$

Данный уровень рентабельности подтверждает экономическую обоснованность реализации проекта и свидетельствует о его высоком потенциале для будущего роста и развития.

7.5 Вывод об экономической целесообразности реализации проектного решения

Проект по разработке программного средства для анимации изображений на основе нейронных сетей продемонстрировал положительные результаты с точки зрения экономической целесообразности. Ожидаемый ROI составляет 286,69%, что указывает на высокую рентабельность и привлекательность инвестиций. Чистая прибыль, которую проект может генерировать ежегодно, составляет около 80 тыс. белорусских рублей. Эти показатели подтверждают, что проект обладает значительным потенциалом для получения прибыли в краткосрочной и среднесрочной перспективе.

Для привлечения большего числа пользователей, проект может предложить базовую бесплатную версию программного средства с ограниченными функциями. Это создаст дополнительную привлекательность продукта, позволит пользователям ознакомиться с функционалом и, возможно, приведет к конверсии в платные подписки для получения расширенных возможностей.

Но несмотря на привлекательность продукта, рынок генерации видео уже насыщен конкурентами, предлагающими аналогичные решения. Это создает угрозу для захвата значительной доли рынка, особенно если продукт не будет достаточно уникален или не оправдает ожидания пользователей в плане качества.

Также стоит учесть, что реализация нейронных сетей для генерации видео на основе изображений и текста – это сложная задача. Возможны трудности с вычислительными ресурсами, а также с оптимизацией работы нейросетевых моделей для качественной и быстрой генерации контента.

С учетом правильного подхода к улучшению качества продукта, актуальности для целевых пользователей и эффективному маркетингу, проект имеет все шансы на успех и получение прибыли в короткие сроки.

ЗАКЛЮЧЕНИЕ

В ходе прохождения преддипломной практики была изучена предметная область генеративных моделей, в том числе архитектуры GAN (англ. Generative Adversarial Networks) для создания изображений и видео. Был проведен подробный анализ существующих аналогов, их возможностей и недостатков, что позволило определить оптимальные подходы к реализации проекта.

В процессе выполнения практики были выполнены следующие задачи:

- разработан и реализован модуль обучения генеративной модели для создания изображений и видео;
- продумана архитектура приложения с использованием Streamlit для создания интерактивного пользовательского интерфейса и FastAPI для реализации серверной части;
- настроена система логирования результатов обучения и визуализации данных;
- подготовлено окружение для локального и серверного развертывания проекта;
- проведены эксперименты по обучению модели на различных датасетах, с последующей оценкой качества работы генеративных моделей.

Кроме того, в рамках преддипломной практики были разработаны следующие графические материалы:

- вводный плакат;
- структурная схема системы;
- диаграмма классов, описывающая основные компоненты приложения;
- диаграмма последовательности.

Также был проведен анализ аналогичных решений на рынке, что позволило выделить ключевые преимущества и области для улучшения разработки. Это исследование легло в основу технико-экономического обоснования для возможности реализации проекта на рынке.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Lumen5 [Электронный ресурс]. – Режим доступа : https://lumen5.com.

[2] Synthesia [Электронный ресурс]. – Режим доступа : https://www.synthesia.io.

[3] InVideo [Электронный ресурс]. – Режим доступа : https://www.ourcrowd.com/startup/invideo.

[4] Diffusion Models vs. GANs vs. VAEs: Comparison of Deep Generative Models [Электронный ресурс]. – Режим доступа : https://pub.towardsai.net/diffusion-models-vs-gans-vs-vaes-comparison-of-deep-generative-models-67ab93e0d9ae.

[5] Generative Adversarial Nets [Электронный ресурс]. – Режим доступа : https://arxiv.org/pdf/1406.2661.

[6] Auto-Encoding Variational Bayes [Электронный ресурс]. – Режим доступа : https://arxiv.org/pdf/1312.6114.

[7] What are Diffusion Models? [Электронный ресурс]. – Режим доступа : https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

[8] What is a recurrent neural network? [Электронный ресурс]. – Режим доступа : https://www.ibm.com/think/topics/recurrent-neural-networks.

[9] NeRF [Электронный ресурс]. – Режим доступа : https://www.matthewtancik.com/nerf.

[10] StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks [Электронный ресурс]. – Режим доступа : https://arxiv.org/pdf/1612.03242.

[11] Adversarial Video Generation on Complex Datasets [Электронный ресурс]. – Режим доступа : https://arxiv.org/pdf/1907.06571v2.

[12] Temporal Generative Adversarial Nets with Singular Value Clipping [Электронный ресурс]. – Режим доступа : https://arxiv.org/pdf/1611.06624v3.

[13] Generating Videos with Scene Dynamic [Электронный ресурс]. – Режим доступа : https://arxiv.org/pdf/1609.02612.

[14] MoCoGAN: Decomposing Motion and Content for Video Generation [Электронный ресурс]. – Режим доступа : https://arxiv.org/pdf/1707.04993.

[15] TiVGAN: Text to Image to Video Generation with Step-by-Step Evolutionary Generator [Электронный ресурс]. – Режим доступа : https://arxiv.org/pdf/2009.02018v2.

[16] FutureGAN: Anticipating the Future Frames of Video Sequences using Spatio-Temporal 3d Convolutions in Progressively Growing GANs [Элетронный ресурс]. – Режим доступа : https://arxiv.org/pdf/1810.01325.

[17] Python 3.12 documentation [Электронный ресурс]. – Режим доступа : https://docs.python.org/3.12/.

[18] Самые популярные языки программирования в 2025 году [Электронный ресурс]. – Режим доступа : https://dan-it.com.ua/blog/samye-populjarnye-jazyki-programmirovanija-v-2025-godu/.

[19] TIOBE Index for March 2025 [Электронный ресурс]. – Режим доступа : https://www.tiobe.com/tiobe-index/.

[20] FastAPI [Электронный ресурс]. – Режим доступа : https://fastapi.tiangolo.com.

[21] A Beginners Guide To Streamlit [Электронный ресурс]. – Режим доступа : https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/.

[22] Why and how to use Google Colab [Электронный ресурс]. – Режим доступа : https://www.techtarget.com/searchenterpriseai/tutorial/Why-and-how-to-use-Google-Colab.

[23] A quick introduction to clean architecture [Электронный ресурс]. – Режим доступа : https://www.freecodecamp.org/news/a-quick-introduction-to-clean-architecture-990c014448d2/.

[24] Зарплаты Product-менеджеров [Электронный ресурс]. – Режим доступа : https://geeklink.io/salary-stat/product-menedzher/.

[25] Зарплаты инженеров машинного обучения [Электронный ресурс]. – Режим доступа : https://geeklink.io/salary-stat/machine-learning-engineer/.

[26] Зарплаты Python-разработчиков [Электронный ресурс]. – Режим доступа : https://geeklink.io/salary-stat/python-razrabotchik/.

[27] Зарплаты тестировщиков ПО [Электронный ресурс]. – Режим доступа : https://geeklink.io/salary-stat/testirovshhik-po/.

[28] Invideo Pricing [Электронный ресурс]. – Режим доступа : https://invideo.io/studio-pricing.

[29] Renderforest Pricing[Электронный ресурс]. – Режим доступа : https://www.renderforest.com/ru/subscription.

[30] Wave.video Pricing [Электронный ресурс]. – Режим доступа : https://wave.video/pricing.

[31] Pictory.ai Pricing [Электронный ресурс]. – Режим доступа : https://pictory.ai/pricing.

[32] Synthesys Pricing [Электронный ресурс]. – Режим доступа : https://synthesys.io/pricing/.

[33] AI Studios Pricing [Электронный ресурс]. – Режим доступа : https://www.aistudios.com/pricing.

[34] Vidnoz Pricing [Электронный ресурс]. – Режим доступа : https://www.vidnoz.com/pricing.html.

[35] Pipio.ai Pricing [Электронный ресурс]. – Режим доступа : https://www.pipio.ai/pricing.

[36] Colossyan Pricing [Электронный ресурс]. – Режим доступа : https://www.colossyan.com/pricing.

[37] Vidau Pricing [Электронный ресурс]. – Режим доступа : https://www.vidau.ai/pricing/.

[38] Fliki Pricing [Электронный ресурс]. – Режим доступа : https://fliki.ai/pricing.

[39] Elai Pricing [Электронный ресурс]. – Режим доступа : https://elai.io/pricing/.

Стандарт предприятия. Дипломные проекты (работы). Общие требования. СТП 01-2024 [Электронный ресурс]. – Режим доступа : https://www.bsuir.by/m/12_100229_1_185678.pdf.

ПРИЛОЖЕНИЕ А
(обязательное)

Вводный плакат. Плакат

ПРИЛОЖЕНИЕ Б
(обязательное)

Программное средство для анимации изображений на основе нейронных сетей. Схема структурная

ПРИЛОЖЕНИЕ В
(обязательное)

Программное средство для анимации изображений на основе нейронных сетей. Диаграмма классов. Диаграмма последовательности