Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский Томский политехнический университет»

Школа / филиал	ИЯТШ ТПУ
Обеспечивающее подразделение	ОММФ
Направление подготовки / специальность	01.03.02 Прикладная математика и информатика
Образовательная программа (направленность (профиль))	Прикладная математика в инженерии

ОТЧЕТ ПО ТВОРЧЕСКОМУ ПРОЕКТУ
НА ТЕМУ:

Text to speech: архитектура Tacotron 2 + HiFi-GAN

Выполнил:
Студент группы 0В31 Наливайко Н.Е.

Проверил:
Доцент,к.ф.-м.н Мерзликин Б.С.

Томск 2024

Содержание

Введение

Голос является одним из наиболее распространенных и естественно используемых человеком интерфейсов общения. Благодаря последним достижениям в области технологий голос используется в качестве основного интерфейса в системах искусственного интеллекта. Соответственно, с ростом спроса на общение людей с машинами активно изучается технология, которая синтезирует естественную речь, подобную человеческой [2].

Системы преобразования текста в речь (TTS) достигли значительных успехов, но создание естественного звучания человеческого голоса остается сложной задачей, несмотря на десятилетия исследований [5]. Со временем в этой области начали доминировать различные методы. Конкатенативный синтез с выбором единиц измерения, процесс объединения небольших блоков предварительно записанных сигналов вместе, был самым современным на протяжении многих лет. Статистический параметрический синтез речи, который непосредственно генерирует плавные траектории речевых признаков для последующего синтеза вокодером, решая многие из проблем, которыми обладал конкатенативный синтез. Однако звук, воспроизводимый этими системами, часто звучит приглушенно и неестественно по сравнению с человеческой речью [1]. Чтобы преодолеть это, для TTS были предложены решения, основанные на глубоком обучении (DL), но для них требуется большой объем обучающих данных [5].

В этой работе в качестве основной архитектуры акустической модели выбран Tacotron 2. Он состоит из рекуррентной сети последовательного предсказания признаков, которая отображает вложения символов в спектрограммы масштаба мел, за которыми следует модифицированная модель WaveNet, действующая как вокодер для синтеза сигналов во временной области из этих спектрограмм. Модель способна воспроизводить естественные интонации, ударения и паузы благодаря использованию механизма внимания и рекуррентных слоёв, что делает синтез более похожим на человеческую речь [1]. В качестве главного вокодера системы использован HiFi–GAN. Эта технология умеет моделировать периодические паттерны звука, что важно для создания реалисичного вывода. Для этого структура модели выстроена в виде нескольких субдискриминаторов, каждый из которых получает только определенные периодические части необработанных сигналов. Поэтому модель способна преобразовывать мел-спектрограммы в аудиосигнал в реальном времени [2].

Цель работы: построение text to speech модели.

Задачи работы:

1. Изучение действующих подходов для синтеза аудиосигналов.
2. Подбор архитектуры на основе актуальных работ.
3. Построение text to speech модели.
   3.1. Выбор датасета.
   3.2. Обучение.
4. Анализ полученной модели.

Теоретический раздел
1. Введение в TTS
1.1. Не нейросетевые подходы

Технология синтеза речи появилась еще в 18 веке, когда австро-венгерский ученый-механик Вольфганг фон Кемпелен создал «говорящую машину». Это была система из мехов, трубок и мембран, которая воспроизводила звуки, напоминающие речь. В 1937 году инженер Гомер Дадли из Bell Labs разработал Voder — первое электронное устройство для синтеза речи на основе электрических сигналов. К 1960-м годам TTS начала переходить в цифровую эпоху. В MIT сделали одну из первых компьютерных систем — DECtalk, основанную на цифровых методах. Однако из-за ограниченных вычислительных мощностей синтезированная речь звучала искусственно. Лишь в 1990-х, с появлением более производительных компьютеров, получилось улучшить качество голосов и сделать их приближенными к человеческой речи [11].

Конкатенативный синтез. Этот метод основан на предварительной записи коротких аудио-фрагментов, которые затем объединяются для создания связной речи. Она получается очень чистая и ясная, но абсолютно лишена эмоциональной и интонационной составляющих, то есть звучит неестественно. А всё потому, что невозможно получить аудиозапись всех возможных слов, произносимых во всех возможных сочетаниях эмоций и просодии. Конкатенативные системы требуют огромных баз данных и жесткого кодирования комбинаций для формирования слов. Разработка надежной системы занимает очень много времени [11].

Параметрический синтез речи. Этот статистический метод, исследует саму природу данных. Он генерирует речь с помощью комбинирования таких параметров, как частота, спектр амплитуд и т.д. Этапы параметрического синтеза:

1. Из текста извлекаются лингвистические признаки: фонемы, продолжительность и т.д.
2. Извлекаются признаки, которые представляют соответсвтующий речевой сигнал: кепстр, линейная спектрограмма, мел-спектрограмма, после признаки подаются в вокодер.
3. Вокодер выполняет преобразования для генерирования звуковой волны, оценивая параметры речи

Если мы можем аппроксимировать параметры, которые определяют речь на каждой её единице, тогда мы сможем создать параметрическую модель. Параметрический синтез требует значительно меньше данных и тяжелой работы, нежели конкатенативные системы.

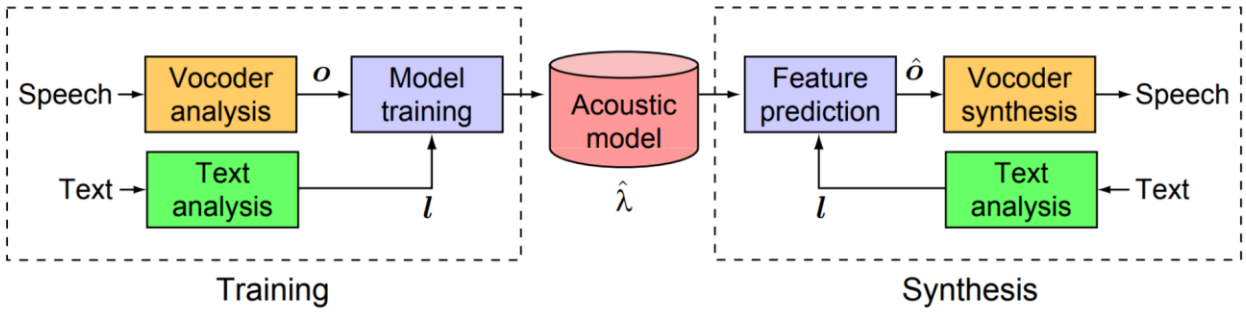


Рис. 1. Архитектура параметрического синтеза [7]

Не нейросетевые подходы к генерации аудио, имеют как преимущества, так и недостатки. Конкатенативный синтез обеспечивает высокое качество звучания благодаря использованию реальных аудиофрагментов, но требует огромной базы данных и не способен гибко передавать эмоции или новые интонации. Параметрический синтез более гибкий и требует меньше данных, однако часто звучит неестественно и синтетически. Оба метода уступают современным

нейросетевым подходам в передаче выразительности, естественности и универсальности речи [Promt_1].

1.2. Нейросетевые подходы

TTS на основе нейронной сети (end-to-end) за последние годы значительно улучшило качество синтезированной речи. Как правило, задача делится на более управляемые подзадачи с использованием генератора акустических характеристик и нейронного вокодера. В этой двухэтапной системе генератор акустических признаков сначала генерирует акустический признак из входного текста, а затем нейронный вокодер синтезирует необработанную форму сигнала из акустического признака. Эти модели обучаются отдельно, а затем объединяются для вывода. Генератор акустических характеристик может быть авторегрессирующим и основанный на механизме внимания для неявного выравнивания речи и текста, или он может быть неавторегрессионным для эффективного параллельного вывода и может быть информирован о продолжительности для обеспечения устойчивости к ошибкам синтеза. Также существует множество исследований по нейронному вокодеру, и некоторые из известных, широко используемых кодеров включают, основанный на нормализации потока и основанный на генеративной состязательной сети (GAN) [12].

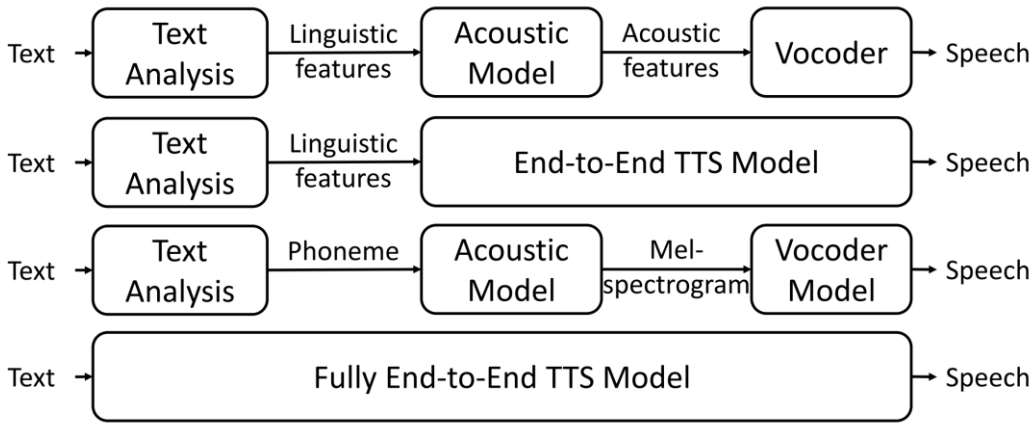


Рис. 2. End-to-end нейронная сеть [7]

Текст, который подается в модель, нужно привести в нормальный вид, то есть раскрыть числа и сокращения, а также проставить ударения во всех словах (для русского языка желательно разрешить омографы). Акустическая модель (энкодер) предсказывает мел-спектрограммы на основе токенов, полученных после нормализации. Это сжатое представление аудио подается в вокодер, который может быть представлен как параметрическими методами синтеза речи, так и нейросетью (WaveNet, SampleRNN, Diffusion-based model).

Рассмотрим некоторые end-to-end TTS. WaveNet — это автоагрессивная модель, основанная на сверточной архитектуре, которая генерирует аудио по одному семплу за раз, обеспечивая высокое качество, но с большой вычислительной затратой. FastSpeech — это неавторегрессионная модель, построенная для ыстрого синтеза речи, использующая duration predictor и transformer-базированную архитектуру для генерации спектрограмм параллельно, значительно ускоряя вывод по сравнению с Tacotron. ClariNet сочетает в себе неавторегрессионный генератор спектрограмм с вокодером, основанным на flow-базированных моделях, для более стабильного и реалистичного синтеза речи.

Нейросетевые подходы обеспечивают более естественное и выразительное звучание речи, позволяют моделировать сложные зависимости между текстом и аудио, поддерживают обучение end-to-end и лучше адаптируются к различным голосам и стилям речи. Однако такие модели требуют большого объема данных для обучения, значительных вычислительных ресурсов, особенно при генерации с автоагрессивными моделями (WaveNet), и могут быть менее устойчивы к ошибкам в предобработке текста или выравнивании аудио и текста [Promt_ 2].

1.3. Мел-спектрограмма

Люди лучше улавливают различия на более низких частотах, чем на более высоких. Более подробно, человек может легко определить разницу между 500 и 1000 Гц, но вряд ли способен определить разницу между 10 000 и 10 500 Гц, даже если расстояние между двумя парами одинаково.Мел-шкала имитирует восприятие звука человеком. Звуки, находящиеся на одинаковом расстоянии друг от друга, воспринимаются как равноудаленные для человека. Преобразование шкалы Гц в мел-шкалу вычисляется с помощью уравнений [6]:

$$M = 1127 \times \log(1 + f/700)$$

Мел-спектрограммы строяться по следующему принципу: к сигналу применить оконное преобразование Фурье (STFT), перевести в мел-шкалу. Сами спектрограммы показывают график зависимости плотности мощности сигнала от времени и частоты:

$$E(\tau, \omega) = |F(\tau, \omega)|^2$$

, где $F(\tau, \omega)$ – оконное преобразование Фурье.

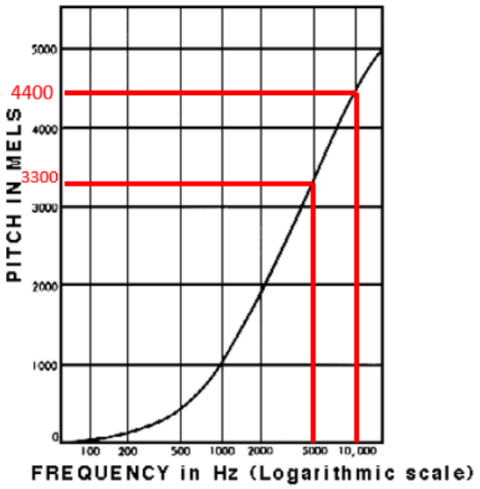


Рис. 3. Зависимость Гц от мел-шкалы [6]

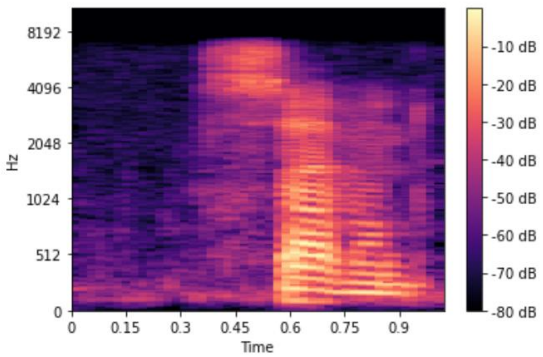


Рис. 4. Мел-спектрограмма [6]

2. Архитектура

Стандартный подход для решение text to speech задач, представлен архитектурой Tacatron 2 и WaveNet. Однако основным недостатков звуковых вокодеров, связанных с использованием данной структуры, является то, что они могут генерировать только один речевой пример за один проход. WaveNet это модель авторегрессии, которая использует предыдущие сэмплы для генерации следующих, что может может увеличить время обработки. Заметных улучшений в этой области пока не найдено, по этой причине в качестве вокодера, будет использоваться HiFi-GAN. Обучение по MSE loss.

2.1. Tacotron 2

Система состоит из рекуррентной сети последовательного предсказания признаков, которая отображает эмбеддинги символов в мел-спектрограммы, за которыми следует модифицированная модель WaveNet, действующая как вокодер для синтеза сигналов во временной области из этих спектрограмм. Оригинальная версия Tacotron использует алгоритм Гриффина-Лима для оценки фазы, за которым следует обратное кратковременное преобразование Фурье (STFT).

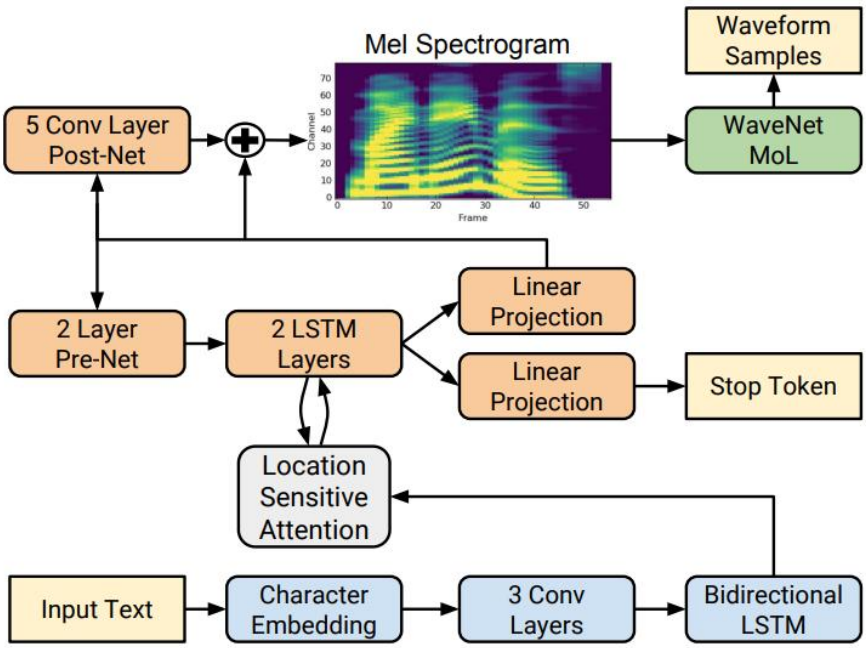


Рис. 5. Архитектура Tacotron 2 [1]

Спектрограммы, вычисляемые по STFT используют размер кадра 50 мс, 12.5 мс для перехода по кадру и функцию окна Ханна. Полученную величину преобразуют в мел-спектрограмму, используя 80-канальный набор фильтров с частотой от 125 Гц до 7.6 кГц, с последующим логарифмическим сжатием динамического диапозона. Перед логарифмическим сжатием выходные значения набора фильтров ограничены до минимального значения 0.01.

2.1.1. Encoder

Энкодер преобразует последовательность символов в представление скрытого признака, которое используется декодером для прогнозирования спектрограммы. Входные символы представляются с использованием выученного 512-мерного эмбеддинга, которые

пропускаются через стопку из 3 сверточных слоев, каждый из которых содержит 512 фильтров с формой 5 × 1, т.е. где каждый фильтр охватывает 5 символов, с последующей нормализацией батчей и ReLu. Выходные данные конечного сверточного слоя передаются в единый двунаправленный LSTM, содержащий 512-мерный скрытые представления (по 128 на каждый слой), для генерации обьектов [1].

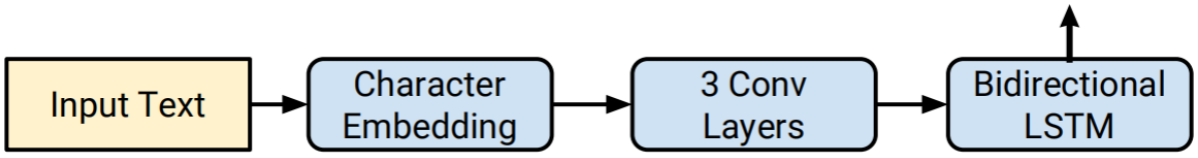


Рис. 6. Архитектура Encoder-слоя [1]

2.1.2. Decoder + Attention Mechanism

Декодер представляет собой рекуррентную нейронную сеть с авторегрессией, которая предсказывает мел-спектрограмму на основе кодированной входной последовательности по одному кадру за раз. Предсказание, полученное на предыдущем временном шаге, сначала проходит через небольшую предварительную сеть, содержащую 2 полностью связанных слоя из 256 скрытых блоков ReLU.cТекущий выходной сигнал и вектор контекста внимания объединяются и передаются через стек из 2 однонаправленных слоев LSTM с 1024 единицами [3]. Объединение выходных данных LSTM и вектора контекста внимания проецируется посредством линейного преобразования для прогнозирования целевого кадра спектрограммы. Наконец, предсказанная мел-спектрограмма пропускается через 5-слойную сверточную сетку, которая предсказывает остаточную величину, добавляемую к прогнозу для улучшения общей реконструкции. Каждый слой конечной сети состоит из 512 фильтров размером 5 × 1 с нормализацией батчей, за которой следует активация tanh на всех слоях, кроме предыдущего [1].

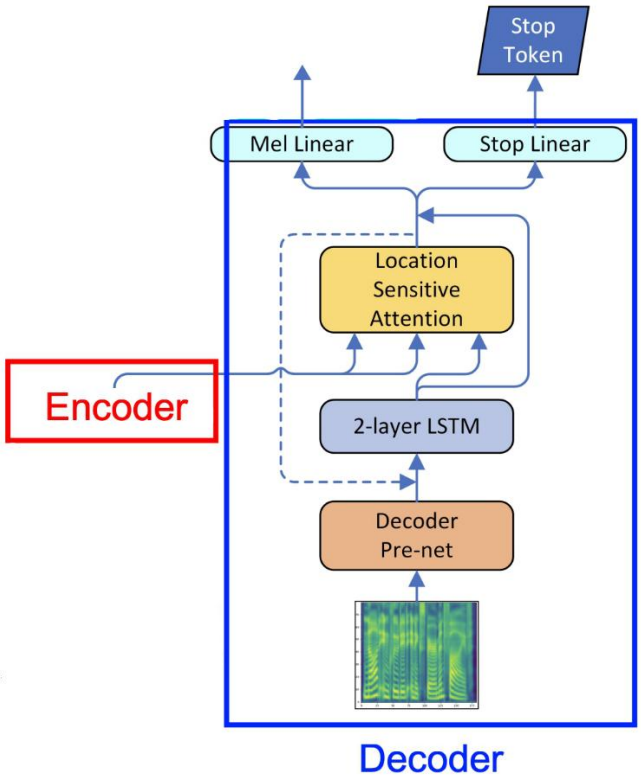


Рис. 7. Архитектура Decoder-Attention слоев [10]

2.2. HiFi-GAN

HiFi-GAN состоит из одного генератора и двух дискриминаторов: многомасштабного и многопериодного. Генератор и дискриминаторы обучаются в разных режимах, а также с двумя дополнительными потерями для повышения стабильности обучения и производительности модели [2].

2.2.1. Generator

Генератор представляет собой полностью сверточную нейронную сеть. Он использует мел-спектрограмму в качестве входных данных и выполняет ее выборку с помощью транспонирования сверток до тех пор, пока длина выходной последовательности не будет соответствовать временному разрешению необработанных сигналов.

Модуль multi-receptive field fusion (MRF) параллельно отслеживает шаблоны различной длины. В частности, модуль MRF возвращает сумму выходных данных из нескольких остаточных блоков. Для каждого остаточного блока выбираются различные размеры ядра и степени разбавления, чтобы сформировать различные схемы поля восприятия. Настраеваемые гиперпараметры: размерность скрытого слоя h_u, размер ядра k_u транспонированных сверток, размеры ядра k_r и расширенная скорость D_r [2].

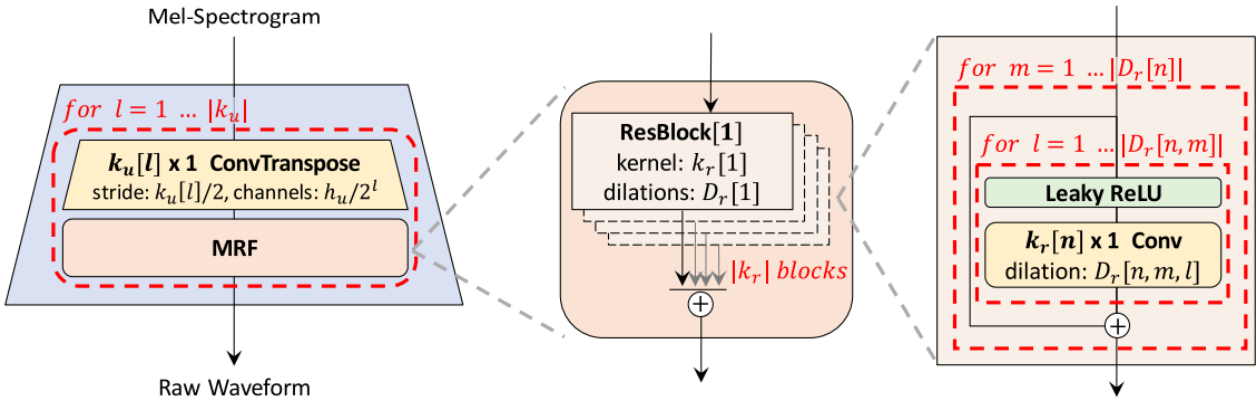


Рис. 8. Архитектура генератора [2]

2.2.2. Discriminator

Поскольку речевой звук состоит из синусоидальных сигналов с различными периодами, необходимо идентифицировать различные периодические паттерны, лежащие в основе аудиоданных. С этой целью предложен многопериодический дискриминатор (MPD), состоящий из нескольких субдискриминаторов, каждый из которых обрабатывает часть периодических сигналов входного аудиосигнала. Кроме того, для выявления последовательных паттернов и долгосрочных зависимостей используется многомасштабный дискриминатор (MSD), который последовательно оценивает звуковые образцы на разных уровнях.

Multi-Period Discriminator. Представляет собой смесь субдискриминаторов, каждый из которых принимает только равноотстоящие выборки входного аудиосигнала; интервал задается как период p. Субдискриминаторы предназначены для выделения различных неявных структур, отличающихся друг от друга, при просмотре разных частей входного аудиосигнала. Период следует брать из: [2, 3, 5, 7, 11], чтобы максимально избежать совпадений. Сначала преобразуем 1D-аудио длиной T в 2D-данные высотой T/p и шириной p, а затем применяем

2D-свертки к измененным данным. На каждом сверточном слое MPD ограничиваем размер ядра по оси равным 1, чтобы обрабатывать периодические выборки независимо. Каждый субдискриминатор представляет собой стопку сверточных слоев с шагами и активацией Leaky ReLU. Впоследствии к MPD применяется нормализация веса Путем преобразования входного аудиосигнала в 2D-данные вместо выборки периодических звуковых сигналов градиенты из MPD могут быть переданы на все временные этапы входного аудиосигнала [2].

Multi-Scale Discriminator. Представляет собой смесь трех субдискриминаторов, работающих с разными входными масштабами: необработанный звук, усредненный объединенный звук ×2 и усредненный объединенный звук ×4. Каждый из субдискриминаторов в MSD представляет собой набор ступенчатых и сгруппированных сверточных слоев с активацией Leaky ReLU. Применяется нормализация веса, за исключением первого субдискриминатора, который работает с
необработанным звуком. Вместо этого применяется спектральная нормализация, которая стабилизирует тренировку.

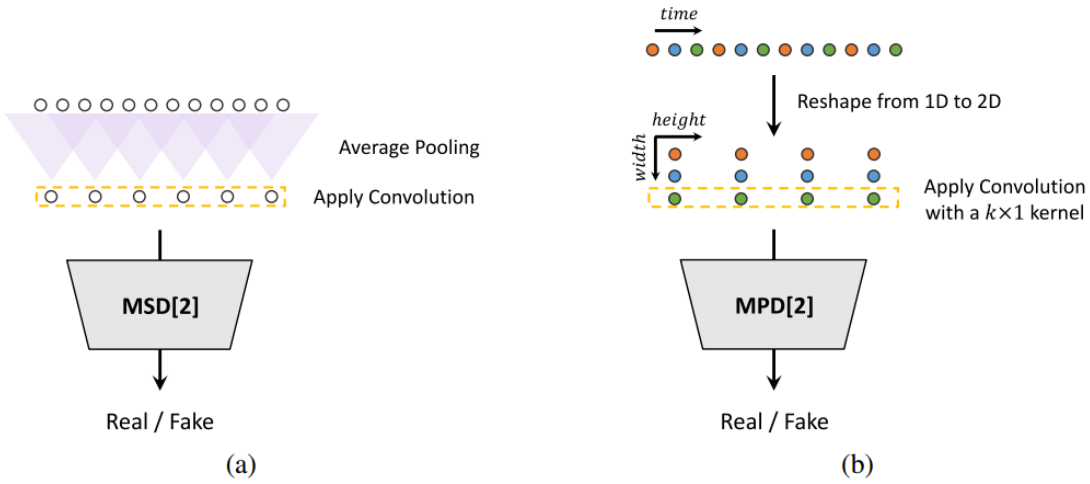


Рис. 9. Архитектура дискриминатора [2]

2.2.3. Train

Функция потерь, на которой будет происходить обучение HiFi-GAN, выглядит следующим образом [2]

$$\mathcal{L}_G = \mathcal{L}_{Adv}(G,D) + \lambda_{fm}\mathcal{L}_{FM}(G,D) + \lambda_{mel}\mathcal{L}_{Mel}(G)$$
$$\mathcal{L}_D = \mathcal{L}_{Adv}(D,G)$$

, где G – генератор, D – дискриминатор, $\lambda_{fm} = 2, \lambda_{mel} = 45$, и соответствующие лоссы равны:

$$\mathcal{L}_{Adv}(G,D) = \mathbb{E}_s\left[\left(D\left(G(s)\right) - 1\right)^2\right]$$
$$\mathcal{L}_{Adv}(D,G) = \mathbb{E}_{(x,s)}\left[(D(x)-1)^2 + D\left(G(s)\right)^2\right]$$
$$\mathcal{L}_{FM}(G,D) = \mathbb{E}_{(x,s)}\left[\sum_{i=1}^{T}\frac{1}{N_i}\left\|D^i(x) - D^i\left(G(s)\right)\right\|_1\right]$$
$$\mathcal{L}_{Mel}(G) = \mathbb{E}_{(x,s)}[\|\phi(x) - \phi(G(s))\|_1].$$

Практический раздел

3. Построение модели

3.1. Датасет

Работа с русскоязычными датасетами для TTS сталкивается с рядом сложностей: русский язык обладает сложной морфологией, богатой флексией и акцентологическими особенностями. Корректная лемматизация и токенизация требуют расширенных правил, а омографы — слова, пишущиеся одинаково, но имеющие разное произношение в зависимости от контекста — требуют ручной разметки или специальных алгоритмов для расстановки ударений. Эти факторы затрудняют автоматическую предобработку текста и делают обучение моделей более трудоемким. В результате, подбор весов для архитектуры, такой как Tacotron 2 + HiFi-GAN, на русском языке требует либо готового размеченного корпуса, либо значительных вычислительных мощностей и большого количества времени для последовательного обучения модели с нуля.

В связи с этим был выбран англоязычный датасет LJSpeech — один из наиболее популярных и открытых корпусов для обучения TTS. Он содержит 13 100 аудиофайлов общей длительностью около 24 часов, записанных одной дикторшей, с соответствующими текстами из книг и статей. Аудио в формате WAV, частота дискретизации 22,050 Гц, выровненные и очищенные данные. Для улучшения качества звука используется предобученный вокодер HiFi-GAN, веса которого доступны в нескольких вариантах (V1, V2, V3) в зависимости от типа акустических признаков и модели [8]. Эти веса позволяют преобразовывать мел-спектрограммы в высококачественный аудиосигнал без необходимости обучать вокодер с нуля [Promt_ 3].

3.2. Обучение

С предобученной моделью Tacotron 2 возникли сложности: компания NVIDIA, ранее предоставлявшая открытый доступ к весам своей реализации, закрыла возможность их свободного использования [9]. Это делает невозможным дообучение (fine-tuning) модели на собственных данных и существенно ограничивает применение архитектуры в практических задачах. Единственным доступным вариантом остается полное обучение модели с нуля на платформе Google Colab, где доступ к GPU ограничен по времени 4 часами. В таких условиях модель может не достичь высокого качества генерации речи, особенно без расширенного вычислительного ресурса. В то же время для HiFi-GAN была использована предобученная модель, обученная на LJSpeech, что значительно упрощает задачу синтеза аудиосигнала и позволяет получить качественный вокодер без необходимости его повторного обучения [Promt_4].

3.3. Анализ полученной модели

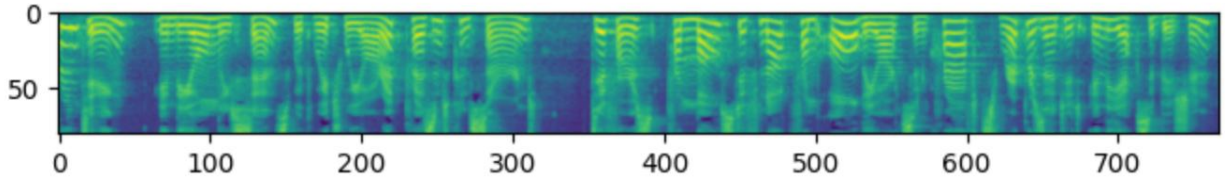


Рис. 10. Мел-спектрограмма на исходных данных

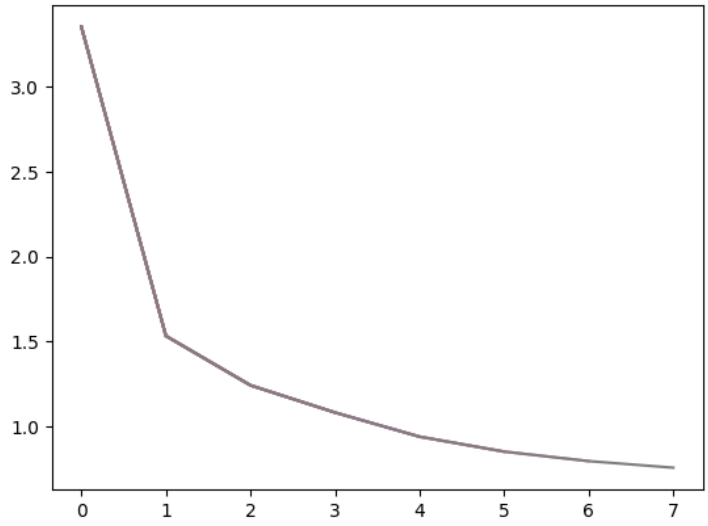


Рис. 11. Кривая обучения

Полученная модель Tacotron 2 показала минимальное значение функции потерь (loss) на уровне 0.7567, что свидетельствует о базовой способности модели к генерации мел-спектрограмм, приближенных к оригинальным. Однако такой результат указывает на то, что модель ещё далека от достижения качественного синтеза речи, сопоставимого с реальной аудиозаписью. Для повышения точности необходима дообучающая фаза на существенно большем количестве эпох, что требует увеличенных вычислительных мощностей и более стабильной среды обучения. В условиях ограниченных ресурсов модель остаётся обученной лишь частично, и её генеративные возможности существенно уступают эталонным реализациям, обученным на мощных GPU-инстансах с большим числом итераций [Promt_ 5].

Реализация модели находится в Приложении. Там же описаны все необходимые зависимости и репозитории, которые будут нужны для воспроизводимости кода.

Заключение

В ходе данной работы была реализована система синтеза речи на основе нейросетевой архитектуры Tacotron 2 с использованием предобученного вокодера HiFi-GAN. Удалось построить полный пайплайн от текстового входа до генерации мел-спектрограмм и последующего преобразования их в аудиосигнал. Несмотря на ограниченные вычислительные ресурсы, обучение модели продемонстрировало базовую сходимость и способность к воспроизведению ключевых акустических признаков речи. Работа позволила закрепить практические навыки в области обработки речи, включая предобработку текстов, построение мел-спектрограмм, обучение seq2seq-моделей и использование GAN-вокодеров.

Полученные результаты могут служить основой для дальнейших экспериментов и оптимизаций. В перспективе можно улучшить модель за счёт увеличения обучающей выборки, использования более мощных GPU-инстансов и применения продвинутых архитектур, таких как FastSpeech 2 или diffusion-based TTS. Также перспективным направлением может стать перенос технологии на другие языки, включая русскоязычные TTS-системы с открытыми корпусами данных. На прикладном уровне подобные системы могут использоваться в голосовых ассистентах, озвучивании текстов, системах помощи людям с нарушениями речи и в генеративных продуктах, связанных с мультимедиа [Promt_6].

Список литературы

1. Shen J., Pang R., Weiss R. J., et al. Natural TTS synthesis by conditioning Wavenet on Mel spectrogram predictions // arXiv preprint arXiv:1712.05884. 2017. URL: https://arxiv.org/pdf/1712.05884.
2. Kong J., Kim J., Bae J. HiFi-GAN: Generative adversarial network for efficient and high fidelity speech synthesis // arXiv preprint arXiv:2010.05646. 2020. URL: https://arxiv.org/pdf/2010.05646.
3. Valle R., Shih K. J., Ma J., et al. BigVGAN: A Universal Neural Vocoder with Large-Scale Training // arXiv preprint arXiv:2203.16852. 2022. URL: https://arxiv.org/pdf/2203.16852.
4. Oord A. van den, Dieleman S., Zen H., et al. WaveNet: A generative model for raw audio // arXiv preprint arXiv:1609.03499. 2016. URL: https://arxiv.org/pdf/1409.0473.
5. A Novel End-to-End Turkish Text-to-Speech (TTS) System via Deep Learning [Электронный ресурс] // ResearchGate. URL: https://www.researchgate.net/publication/370109377_A_Novel_End-to-End_Turkish_Text-to-Speech_TTS_System_via_Deep_Learning.
6. Dhanoa N. Mel-Spectrograms and Data Augmentation for Spoken Digit Classification [Электронный ресурс] // ResearchGate. 2023. URL: https://www.researchgate.net/publication/367452919_Mel-Spectrograms_and_Data_Augmentation_for_Spoken_Digit_Classification.
7. Tan X. Neural TTS Synthesis Tutorial [Электронный ресурс] // Microsoft Research. 2021. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2021/07/ISCSLP2021-TTS-Tutorial-Xu-Tan.pdf .
8. HiFi-GAN: High Fidelity Generative Adversarial Network [Электронный ресурс] // GitHub. URL: https://github.com/jik876/hifi-gan.
9. NVIDIA Tacotron2: PyTorch implementation of Natural TTS synthesis by conditioning Wavenet on Mel spectrogram predictions [Электронный ресурс] // GitHub. URL: https://github.com/NVIDIA/tacotron2?tab=readme-ov-file.
10. Anwarvic – GitHub profile and repositories [Электронный ресурс] // GitHub. URL: https://github.com/anwarvic.
11. Журавлёва Л. А., Кузнецов А. Г. Технология синтеза речи в историко-методологическом аспекте [Электронный ресурс] // КиберЛенинка. URL: https://cyberleninka.ru/article/n/tehnologiya-sinteza-rechi-v-istoriko-metodologicheskom-aspekte/viewer.
12. Lim D., Jung S., Kim E. JETS: Jointly Training FastSpeech2 and HiFi-GAN for End-to-End Text-to-Speech // arXiv preprint arXiv:2203.16852. 2022. URL: https://arxiv.org/pdf/2203.16852.

Приложение

Tacotron 2: Encoder

```
class Encoder(nn.Module):
    def __init__(self, vocab_size, embedding_size):
        super().__init__()

        self.embedding_size = embedding_size

        self.embed = nn.Embedding(vocab_size, embedding_size)
        self.conv = nn.Sequential(
            nn.Conv1d(embedding_size, embedding_size, 5, padding = 2,
bias = True),
            nn.BatchNorm1d(embedding_size),
            nn.ReLU(),
            nn.Dropout(),
            nn.Conv1d(embedding_size, embedding_size, 5, padding = 2,
bias = True),
            nn.BatchNorm1d(embedding_size),
            nn.ReLU(),
            nn.Dropout(),
            nn.Conv1d(embedding_size, embedding_size, 5, padding = 2,
bias = True),
            nn.BatchNorm1d(embedding_size),
            nn.ReLU(),
            nn.Dropout()
        )

        for layer in self.conv:
            if isinstance(layer, nn.Conv1d):
                torch.nn.init.xavier_uniform_(layer.weight, gain =
torch.nn.init.calculate_gain('relu'))

        self.bi_lstm = nn.LSTM(embedding_size, embedding_size // 2,
                               batch_first = True, bidirectional =
True)
        self.drop = nn.Dropout(0.1)

    def forward(self, x):
        x = self.embed(x)
        x = x.transpose(1, 2)
        x = self.conv(x)
        x = x.transpose(1, 2)
        x, _ = self.bi_lstm(x)
        return x
```

Tacotron 2: Attention

```
#@title Attention { form-width: "22%" }
class AlignNN(nn.Module):
    def __init__(self, enc_hidden_size, dec_hidden_size):
        '''
        Calculates the alignment of the arguments, representing the
importance of each encoder output.
        Args:
            enc_hidden_size: output size of encoder at step j
            dec_hidden_size: size of previous hidden state of the
decoder
        '''
        super().__init__()
        self.ll_dec_prev_hidden = nn.Linear(dec_hidden_size, 128)
        self.ll_enc_hidden = nn.Linear(enc_hidden_size, 128)
        self.ll_prev_step_att = nn.Linear(32, 128)
```

```
        self.ll_out = nn.Linear(128, 1)

        self.proj1 = None # saves encoder hidden LL output


    def forward(self, dec_prev_hidden, prev_step_att):
        proj2 = self.ll_dec_prev_hidden(dec_prev_hidden) # (BATCH_SIZE,
seq_len = 1, hidden_size = 128)
        proj3 = self.ll_prev_step_att(prev_step_att.transpose(2, 1))
        return self.ll_out(torch.tanh(self.proj1 + proj2 +
proj3)).squeeze(2)

class Attention(nn.Module):
    def __init__(self, enc_hidden_size, dec_hidden_size):
        '''
        Calculates attention (https://arxiv.org/abs/1409.0473)
        Args:
            enc_hidden_size: output size of encoder at step j
            dec_hidden_size: size of previous hidden state of the
decoder
        '''
        super().__init__()
        self.location_att = nn.Conv1d(2, 32, 31, padding = 15)
        self.align_nn = AlignNN(enc_hidden_size, dec_hidden_size)
        self.mask = None


    def forward(self, enc_out, dec_prev_h, prev_step_att):
        # location features
        location_features = self.location_att(prev_step_att)

        # calculate the attention weights
        att_weights = self.align_nn(dec_prev_h, location_features) #
(BATCH_SIZE, seq_len)

        if self.mask is not None:
            att_weights.data.masked_fill_(self.mask, -float('inf'))

        att_weights = torch.softmax(att_weights, dim = 1)
        return att_weights, torch.bmm(att_weights.unsqueeze(1),
enc_out).squeeze(1)
```

Tacotron 2: Decoder

```
#@title Decoder { form-width: "22%" }
# Pieces of code adapted from: https://github.com/NVIDIA/tacotron2
class Decoder(nn.Module):
    def __init__(self, melspec_frame_shape):
        super().__init__()
        self.pre_net = nn.Sequential(
            nn.Linear(melspec_frame_shape, 256),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.Dropout()
        )

        self.lstm_l1 = nn.LSTM(EMBEDDING_SIZE + 256, 1024, 1,
batch_first = True)
        self.att_nn = None # the attention network
        self.lstm_l2 = nn.LSTM(EMBEDDING_SIZE + 1024, 1024, 1,
batch_first = True)
```

```
        self.linear = nn.Linear(EMBEDDING_SIZE + 1024, 80)
        self.stop_linear = nn.Sequential(
            nn.Linear(EMBEDDING_SIZE + 1024, 1),
        )
        self.post_net = nn.Sequential(
            nn.Conv1d(NR_MELS, 512, 5, padding = 2),
            nn.BatchNorm1d(512),
            nn.Tanh(),
            nn.Dropout(),
            nn.Conv1d(512, 512, 5, padding = 2),
            nn.BatchNorm1d(512),
            nn.Tanh(),
            nn.Dropout(),
            nn.Conv1d(512, 512, 5, padding = 2),
            nn.BatchNorm1d(512),
            nn.Tanh(),
            nn.Dropout(),
            nn.Conv1d(512, 512, 5, padding = 2),
            nn.BatchNorm1d(512),
            nn.Tanh(),
            nn.Dropout(),
            nn.Conv1d(512, NR_MELS, 5, padding = 2),
            nn.BatchNorm1d(NR_MELS),
            nn.Dropout()
        )


    def get_mask_from_lengths(self, lengths):
        max_len = torch.max(lengths).item()
        ids = torch.arange(0, max_len,
out=torch.cuda.LongTensor(max_len))
        mask = (ids < lengths.unsqueeze(1)).bool()
        return mask


    def init_states(self, enc_out, trans_lens, batch_size, seq_len,
train_mode = True):
        self.h_n1 = torch.zeros((1, batch_size, 1024), requires_grad =
train_mode).to(DEVICE) # (nr_layers, BATCH_SIZE, hidden_size)
        self.c_n1 = torch.zeros((1, batch_size, 1024), requires_grad =
train_mode).to(DEVICE) # (nr_layers, BATCH_SIZE, hidden_size)
        self.h_n2 = torch.zeros((1, batch_size, 1024), requires_grad =
train_mode).to(DEVICE) # (nr_layers, BATCH_SIZE, hidden_size)
        self.c_n2 = torch.zeros((1, batch_size, 1024), requires_grad =
train_mode).to(DEVICE) # (nr_layers, BATCH_SIZE, hidden_size)

        self.prev_step_att = torch.zeros((batch_size, seq_len),
requires_grad = train_mode).to(DEVICE)
        self.prev_steps_att = torch.zeros((batch_size, seq_len),
requires_grad = train_mode).to(DEVICE)

        self.context_vec = torch.zeros((batch_size, EMBEDDING_SIZE),
requires_grad = train_mode).to(DEVICE)

        self.enc_out = enc_out
        self.att_nn.align_nn.proj1 =
self.att_nn.align_nn.l1_enc_hidden(enc_out)
        if trans_lens != None:
            self.att_nn.mask = ~self.get_mask_from_lengths(trans_lens)

    def forward(self, prev_frame):
        '''
        Args:
            context_vector: the attention context vector for current
step (BATCH_SIZE, EMBEDDING_SIZE)
```

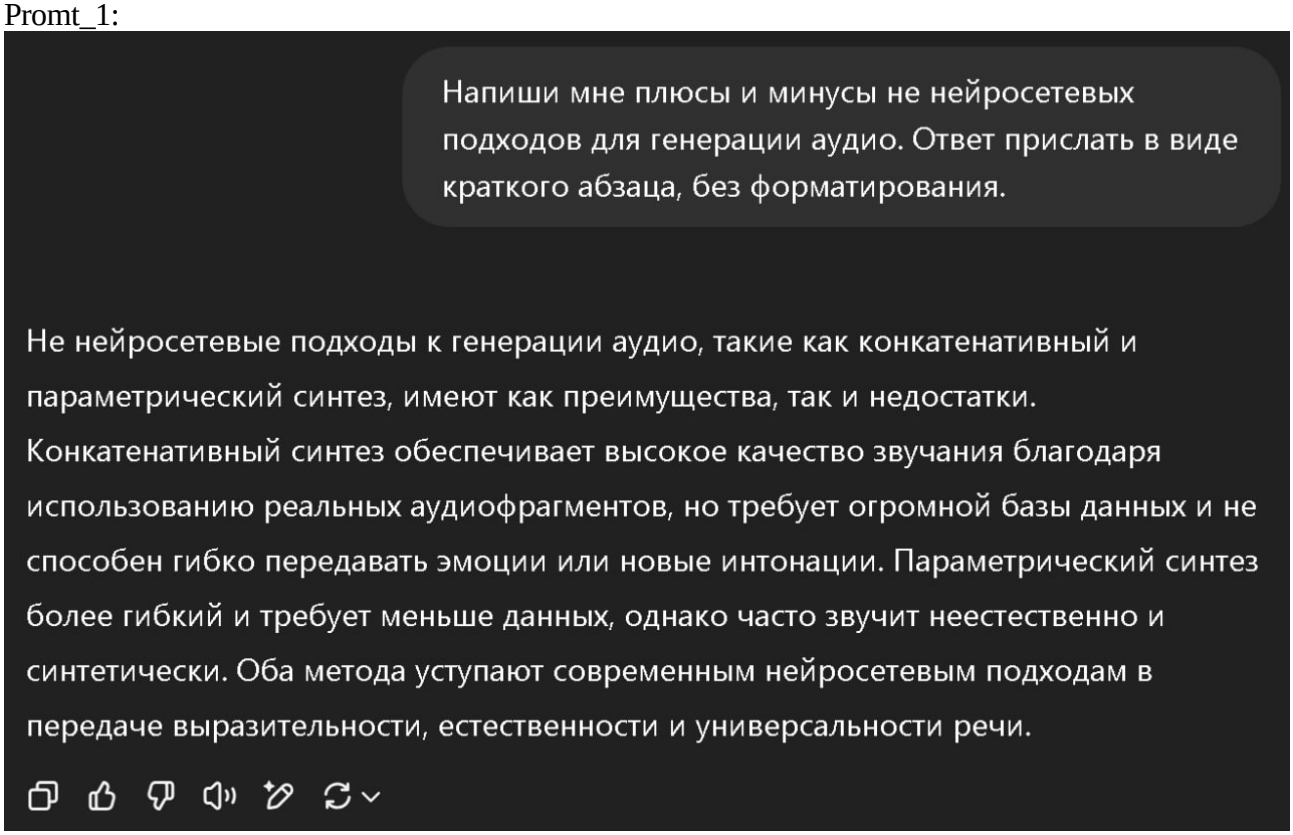
```
            prev_frame: previous step output (frame), using teacher
forcing for training (NR_MELS, BATCH_SIZE)
            h_x: tuple containing last step's hidden and cell states
                h_n shape: (num_layers = 2, BATCH_SIZE, hidden_size =
1024)
                c_n shape: (num_layers = 2, BATCH_SIZE, hidden_size =
1024)
        '''
        prev_frame = prev_frame.transpose(1, 0)

        x = torch.concat([self.context_vec, prev_frame], dim = -1)
        x = x.unsqueeze(1) # LSTM input shape: (batch_size, seq_len =
1, input_len)
        _, (self.h_n1, self.c_n1) = self.lstm_l1(x, (self.h_n1,
self.c_n1))
        self.h_n1 = nn.functional.dropout(self.h_n1, 0.1, True)

        att = torch.cat((self.prev_step_att.unsqueeze(1),
self.prev_steps_att.unsqueeze(1)), dim = 1)
        self.prev_step_att, self.context_vec =
self.att_nn(self.enc_out, self.h_n1.squeeze(0).unsqueeze(1), att)
        self.prev_steps_att += self.prev_step_att

        x = torch.concat([self.h_n1.squeeze(0), self.context_vec], dim
= -1).unsqueeze(1)
        _, (self.h_n2, self.c_n2) = self.lstm_l2(x, (self.h_n2,
self.c_n2))
        self.h_n2 = nn.functional.dropout(self.h_n2, 0.1, True)

        x = torch.concat([self.context_vec, self.h_n2.squeeze(0)], dim
= 1)
        x_out = self.linear(x)
        x_stop = self.stop_linear(x)
        return  x_out, x_stop
```

Dataset:

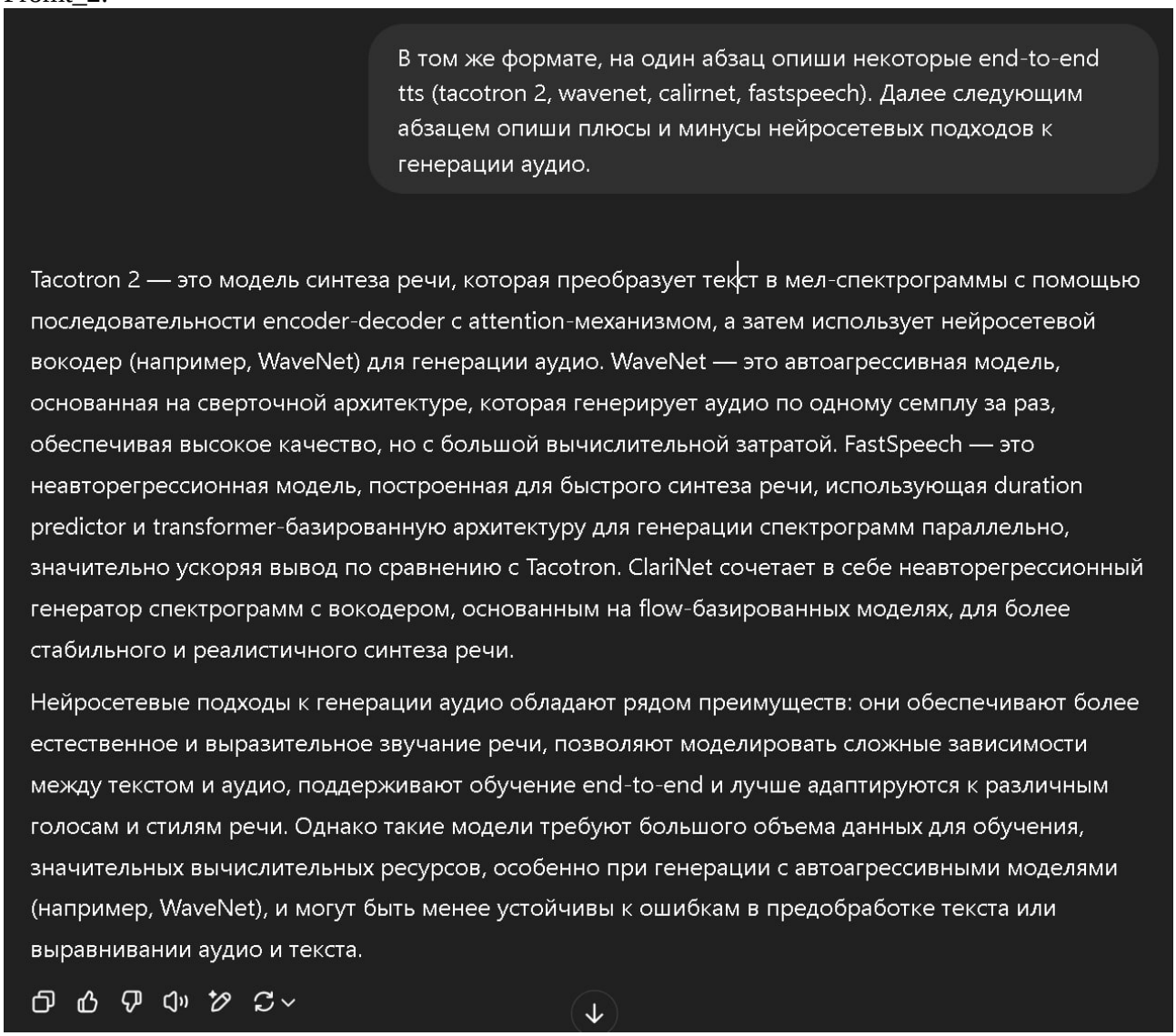
```
#@title Dataset { form-width: "22%" }
class LJSpeech(Dataset):
    def __init__(self, ds_path):
        self.ds = hkl.load(ds_path)
        self.sz = 12000 + 95 + 500 # train, valid, test
        self.ds = self.ds[:self.sz]

    def __len__(self):
        #return len(self.ds)
        return self.sz

    def __getitem__(self, index):
        transcript = torch.LongTensor(self.ds[index][2])
        melspec = torch.FloatTensor(self.ds[index][3])
        stop_token = torch.FloatTensor(self.ds[index][4])
        return transcript, melspec.transpose(1, 0), stop_token #
transponse melspec to pad it in collate

    def min_max(self):
        min, max = 10000, -10000
        for (_, melspec, _, _, _) in train_loader:
            melspec_max = torch.max(melspec)
            if melspec_max > max:
                max = melspec_max

            melspec_min = torch.min(melspec)
            if melspec_min < min:
```

```
                min = melspec_min

        print(min, max)

def collate_pad(batch):
    transcripts, melspecs, stop_tokens = zip(*batch)
    mel_lens = torch.LongTensor([mel.shape[0] for mel in melspecs])
    trans_lens = torch.LongTensor([t.shape[0] for t in transcripts])

    transcripts = nn.utils.rnn.pad_sequence(transcripts, batch_first =
True)
    melspecs = nn.utils.rnn.pad_sequence(melspecs, batch_first = True)
    stop_tokens = nn.utils.rnn.pad_sequence(stop_tokens, batch_first =
True)
    for i, elem in enumerate(stop_tokens):
        elem[mel_lens[i] - 1:] = 1.0
    return transcripts, melspecs.permute(0, 2, 1), stop_tokens,
mel_lens, trans_lens
```

Promt_1:



Promt_2:


В том же формате, на один абзац опиши некоторые end-to-end tts (tacotron 2, wavenet, calirnet, fastspeech). Далее следующим абзацем опиши плюсы и минусы нейросетевых подходов к генерации аудио.
Tacotron 2 — это модель синтеза речи, которая преобразует текст в мел-спектрограммы с помощью последовательности encoder-decoder с attention-механизмом, а затем использует нейросетевой вокодер (например, WaveNet) для генерации аудио. WaveNet — это автоагрессивная модель, основанная на сверточной архитектуре, которая генерирует аудио по одному семплу за раз, обеспечивая высокое качество, но с большой вычислительной затратой. FastSpeech — это неавторегрессионная модель, построенная для быстрого синтеза речи, использующая duration predictor и transformer-базированную архитектуру для генерации спектрограмм параллельно, значительно ускоряя вывод по сравнению с Tacotron. ClariNet сочетает в себе неавторегрессионный генератор спектрограмм с вокодером, основанным на flow-базированных моделях, для более стабильного и реалистичного синтеза речи.
Нейросетевые подходы к генерации аудио обладают рядом преимуществ: они обеспечивают более естественное и выразительное звучание речи, позволяют моделировать сложные зависимости между текстом и аудио, поддерживают обучение end-to-end и лучше адаптируются к различным голосам и стилям речи. Однако такие модели требуют большого объема данных для обучения, значительных вычислительных ресурсов, особенно при генерации с автоагрессивными моделями (например, WaveNet), и могут быть менее устойчивы к ошибкам в предобработке текста или выравнивании аудио и текста.

Promt_3:

В том же формате, напиши небольшой абзац, в котором расскажешь о сложностях работы с датасетами на русском языке (сложный лемминг, омографы). Подведешь итоги что найти веса для данной модели, при заданной архитектуре сложно, а следование обучение потребует больших вычислительных мощностей.
Далее, следующим абзацем расскажи про выбор датасета LJSpeech, немного расскажи о нем (технически). Расскажи про веса для HiFI-GAN.

Работа с русскоязычными датасетами для TTS сталкивается с рядом сложностей: русский язык обладает сложной морфологией, богатой флексией и акцентологическими особенностями. Корректная лемматизация и токенизация требуют расширенных правил, а омографы — слова, пишущиеся одинаково, но имеющие разное произношение в зависимости от контекста — требуют ручной разметки или специальных алгоритмов для расстановки ударений. Эти факторы затрудняют автоматическую предобработку текста и делают обучение моделей более трудоемким. В результате, подбор весов для архитектуры, такой как Tacotron 2 + HiFi-GAN, на русском языке требует либо готового размеченного корпуса, либо значительных вычислительных мощностей и большого количества времени для последовательного обучения модели с нуля.

В связи с этим был выбран англоязычный датасет LJSpeech — один из наиболее популярных и открытых корпусов для обучения TTS. Он содержит 13 100 аудиофайлов общей длительностью около 24 часов, записанных одной дикторшей, с соответствующими текстами из книг и статей. Аудио в формате WAV, частота дискретизации 22,050 Гц, выровненные и очищенные данные. Для улучшения качества звука используется предобученный вокодер HiFi-GAN, веса которого доступны в нескольких вариантах (V1, V2, V3) в зависимости от типа акустических признаков и модели. Эти веса позволяют преобразовывать мел-спектрограммы в высококачественный аудиосигнал без необходимости обучать вокодер с нуля.

Promt_4:

Следующим абзацем, в том ж формате, расскажи о сложностях с предобученной моделью такатрона, nvideo закрыли доступ к весам. Из-за этого модель невозможно fine-tuning, поэтому единственный вариант, это обучать модель на вычислительных мощностях google collab, которые ограничены 4 часами для gpu. В связи с этим качество полученной модели может сильно отличаться от предполагаемого. Для hifi_gan взята уже предобученная модель на датасете ljspeech, что упрощает построение вокодера.

С предобученной моделью Tacotron 2 возникли сложности: компания NVIDIA, ранее предоставлявшая открытый доступ к весам своей реализации, закрыла возможность их свободного использования. Это делает невозможным дообучение (fine-tuning) модели на собственных данных и существенно ограничивает применение архитектуры в практических задачах. Единственным доступным вариантом остается полное обучение модели с нуля на платформе Google Colab, где доступ к GPU ограничен по времени — сессии редко превышают 4 часа. В таких условиях модель может не достичь высокого качества генерации речи, особенно без расширенного вычислительного ресурса. В то же время для HiFi-GAN была использована предобученная модель, обученная на LJSpeech, что значительно упрощает задачу синтеза аудиосигнала и позволяет получить качественный вокодер без необходимости его повторного обучения.

Promt_5:

В формате 1-го абзаца, напиши небольшой анализ по модели. Включи в него информацию о минимальном полученном лоссе : 0.7567, скажи что для более точных результатов требуется большее количество эпох, а следовательно, большее количество вычистельных ресурсов. Добавь свои выкладки.

Полученная модель Tacotron 2 показала минимальное значение функции потерь (loss) на уровне 0.7567, что свидетельствует о базовой способности модели к генерации мел-спектрограмм, приближенных к оригинальным. Однако такой результат указывает на то, что модель ещё далека от достижения качественного синтеза речи, сопоставимого с реальной аудиозаписью. Для повышения точности необходима дообучающая фаза на существенно большем количестве эпох, что требует увеличенных вычислительных мощностей и более стабильной среды обучения. Объём используемого датасета, ограничения по времени на GPU (например, в Google Colab), а также сложность русскоязычного текста (омографы, ударения) становятся критическим узким местом. В условиях ограниченных ресурсов модель остаётся обученной лишь частично, и её генеративные возможности существенно уступают эталонным реализациям, обученным на мощных GPU-инстансах с большим числом итераций.

Воспроизведение модели:

Блокнот с моделью лежит: https://github.com/crdlts?tab=repositories. Там же можно найти веса для построенной модели.

Инструкция:

1. Установить зависимости: requirements.txt.
2. Подгрузить датасет: LJSpeech.
3. Подгрузить веса: taco2_1000_ds_size.pth.
4. Добавить в рабочую директорию файлы: inference_e2e.py, env.py, meldataset.py, models.py, utils.py, config.json, generator_v3.