



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ

«Информатика и системы управления»

КАФЕДРА

«Теоретическая информатика и компьютерные технологии»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
**К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ**  
**НА ТЕМУ:**

***Разработка приложения-собеседника с  
возможностью идентификации пользователей***

Студент ИУ9-82Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

\_\_\_\_\_  
(И.О.Фамилия)

Руководитель ВКР

\_\_\_\_\_  
(Подпись, дата)

\_\_\_\_\_  
(И.О.Фамилия)

Консультант

\_\_\_\_\_  
(Подпись, дата)

\_\_\_\_\_  
(И.О.Фамилия)

Консультант

\_\_\_\_\_  
(Подпись, дата)

\_\_\_\_\_  
(И.О.Фамилия)

Нормоконтролер

\_\_\_\_\_  
(Подпись, дата)

\_\_\_\_\_  
(И.О.Фамилия)

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
ПОСТАНОВКА ЗАДАЧИ.....	5
1 Идентификация пользователей на основе распознавания лиц.....	6
2 Приложения-собеседники.....	7
3 Нейронные сети, используемые для создания приложений- собеседников.....	8
3.1 Рекуррентные нейронные сети.....	10
3.2 Долгая краткосрочная память.....	13
3.3 Закрытый рекуррентный модуль.....	17
3.4 Механизм внимания.....	17
3.5 Нейронные сети трансформеры.....	24
3.6 Модель «последовательность к последовательности».....	27
3.7 Представления двунаправленного энкодера от трансформеров.....	28
3.8 Генеративный предварительно обученный трансформер.....	29
4 Выбор модели нейронной сети для реализации диалога с ботом.....	30
5 Выбор стека технологий.....	31
6 Реализация программы.....	32
6.1 Реализация пользовательского интерфейса приложения.....	32
6.1.1 Реализация интерфейса распознавания лиц и ввода имен.....	32
6.1.2 Реализация интерфейса общения с ботом.....	33
6.2 Реализация базы данных.....	33
6.2.1 Создание модели «сущность-связь», описание сущностей.....	33
6.2.2 Преобразование модели «сущность-связь» в реляционную модель.....	35
6.2.3 Описание таблиц.....	36
6.3 Реализация распознавания лиц.....	37

6.4	Реализация чат-бота.....	39
6.4.1	Выбор данных для обучения.....	39
6.4.2	Предварительная обработка данных.....	39
6.4.3	Реализация выбранной модели нейронной сети.....	40
7	Результаты работы.....	44
7.1	Результаты обучения нейронной сети.....	44
7.2	Пользовательский интерфейс приложения.....	45
7.3	Распознавание лиц и ввод имен.....	47
7.4	Тестирование проведения диалога с ботом.....	50
	ЗАКЛЮЧЕНИЕ.....	53
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	54
	ПРИЛОЖЕНИЕ А.....	56

## **ВВЕДЕНИЕ**

Разработка приложений-собеседников стала одним из актуальных направлений в сфере информационных технологий в современном мире. Технологии искусственного интеллекта, машинного обучения и обработки естественного языка стремительно развиваются, обеспечивая возможность создания умных виртуальных ассистентов, способных взаимодействовать с пользователями.

Приложения-собеседники представляют собой инновационные решения, способные отвечать на вопросы пользователей, поддерживать разговор, помогать в решении задач и обеспечивать персонализированный опыт. Их разработка требует понимания потребностей пользователей, а также технических аспектов в области искусственного интеллекта.

В данной выпускной квалификационной работе рассматривается способ идентификации пользователей на основе распознавания лиц, различные виды моделей нейронных сетей, используемые для создания приложений-собеседников и реализация приложения-собеседника с возможностью идентификации пользователей.

## ПОСТАНОВКА ЗАДАЧИ

Целью выпускной квалификационной работы является реализация приложения-собеседника с возможностью идентификации пользователей при помощи распознавания лиц.

Для достижения поставленной цели необходимо решить следующие задачи:

- рассмотреть способ идентификации пользователей на основе распознавания лиц;
- рассмотреть различные виды нейронных сетей, используемых для создания приложений-собеседников;
- реализовать пользовательский интерфейс приложения – отображение видеопотока с веб-камеры, интерфейс для распознавания лиц и ввода имен, а также интерфейс для общения с ботом;
- реализовать базу данных для хранения информации, необходимой для работы приложения;
- реализовать распознавание лиц и возможность ввода имен;
- реализовать нейронную сеть для работы чат-бота;
- реализовать возможность общения с ботом в пользовательском интерфейсе;
- провести тестирование приложения.

## **1 Идентификация пользователей на основе распознавания лиц**

Распознавание лиц – это способ идентификации или подтверждения личности человека по его лицу. Систему распознавания лиц можно использовать для идентификации людей на фотографиях, видео или в режиме реального времени.

Рассмотрим основные принципы работы таких систем:

1. **Захват изображения лица:** изображение лица может быть получено с помощью камеры или другого устройства, способного захватывать изображения.
2. **Предварительная обработка:** изображение лица может быть подвергнуто предварительной обработке для улучшения качества изображения, удаления шумов и нормализации освещения.
3. **Извлечение особенностей:** для распознавания лиц изображение лица анализируется с целью извлечения уникальных особенностей, таких как расположение глаз, носа, рта, форма лица и другие характеристики.
4. **Создание шаблона лица:** на основе извлеченных особенностей создается уникальный шаблон лица, который представляет собой числовое представление характеристик лица.
5. **Сравнение и идентификация:** полученный шаблон лица сравнивается с заранее сохраненными шаблонами в базе данных для идентификации или верификации личности.
6. **Принятие решения:** на основе результатов сравнения система принимает решение о том, соответствует ли изображение лица зарегистрированной личности или нет.
7. **Обработка ошибок и обучение:** система может быть обучена улучшать свою точность распознавания лиц путем адаптации к новым образцам

лиц и учета различных условий освещения, углов обзора и других параметров.

8. Использование искусственных нейронных сетей: многие системы используют алгоритмы глубокого обучения, такие, как сверточные нейронные сети, для обработки изображений и извлечения признаков, что позволяет им достигать высокой точности распознавания лиц.

## **2 Приложения-собеседники**

Приложения-собеседники — это инновационные приложения, разработанные для ведения разговоров с пользователями на естественном языке. Они обычно основаны на искусственном интеллекте и машинном обучении, что позволяет им постоянно улучшать свои навыки коммуникации. В последние годы приложения-собеседники приобрели огромную популярность и нашли широкое применение в различных областях.

Рассмотрим развитие приложений-собеседников:

1. Искусственный интеллект: с развитием технологий искусственного интеллекта, приложения-собеседники стали способны понимать и анализировать человеческую речь с высокой точностью.
2. Обработка естественного языка (NLP): приложения-собеседники используют технологии обработки естественного языка для понимания контекста разговора и формирования своих ответов.
3. Машинное обучение: благодаря машинному обучению они способны улучшать свои навыки коммуникации на основе обратной связи и опыта общения с пользователями.

Приложения-собеседники имеют следующее влияние на общество и бизнес:

1. Клиентские сервисы: многие компании используют приложения-собеседники для автоматизации обработки запросов клиентов и предоставления быстрой поддержки.

2. **Здравоохранение:** в медицинской сфере приложения-собеседники помогают пациентам получать информацию о состоянии здоровья, предупреждать заболевания и следить за режимом приема лекарств.
3. **Образование:** в сфере образования они могут быть использованы для обучения и помощи студентам в освоении учебного материала.

Приложения-собеседники имеют следующие преимущества:

1. **Эффективность:** приложения-собеседники способны обрабатывать большой объем запросов и обеспечивать постоянную доступность.
2. **Персонализация:** способность приложений-собеседников адаптироваться к индивидуальным потребностям пользователей.
3. **Экономия ресурсов:** автоматизация коммуникаций позволяет сократить затраты на управление клиентским сервисом.

При разработке приложений-собеседников необходимо учитывать следующие требования:

1. **Качество обслуживания:** необходимость внимательного контроля качества и точности ответов приложений-собеседников.
2. **Приватность данных:** защита конфиденциальности информации, передаваемой через приложения-собеседники.

Приложения-собеседники представляют собой мощный инструмент современных технологий, значительно изменяющий образ взаимодействия между человеком и компьютером. Они оказывают большое влияние на бизнес, образование, здравоохранение и другие сферы жизни общества.

### **3 Нейронные сети, используемые для создания приложений-собеседников**

В последнее время нейронные сети стали ключевым инструментом для создания приложений-собеседников. Различные типы нейронных сетей использовались для развития таких приложений. Рассмотрим основные из них и их роли в создании приложений-собеседников:

1. **Рекуррентные нейронные сети (RNN):** рекуррентные нейронные сети широко используются для создания приложений-собеседников в силу



своей способности работать с последовательностями данных. Однако, у RNN есть ограничения в обработке длинных зависимостей из-за проблемы затухания или взрыва градиентов.

2. LSTM (Long Short-Term Memory): LSTM является одним из типов RNN. LSTM используется для решения проблемы затухания градиентов, позволяет сохранять информацию на протяжении длительных временных интервалов. LSTM является эффективным для моделей, где важны долгосрочные зависимости, таких как приложения-собеседники.
3. GRU (Gated Recurrent Unit): модель GRU основана на тех же принципах, что и LSTM, но использует меньше фильтров и операций для вычисления нового скрытого состояния. GRU также предназначен для работы с последовательностями и может быть использован в разработке приложений-собеседников.
4. Механизм внимания: механизм внимания позволяет нейронной сети фокусироваться на определенных частях входных данных в зависимости от контекста, что приводит к улучшению качества ответов в приложениях-собеседниках, делая их более точными и информативными.
5. Нейронные сети Transformer: трансформеры – это модели нейронных сетей, которые обеспечивают параллельную обработку данных без необходимости учета порядка слов. Модели Transformer хорошо подходят для создания приложений-собеседников, так как могут распараллеливаться и способны обрабатывать большие объемы текста.
6. Модель sequence to sequence: модель sequence to sequence (seq2seq) применяется для преобразования последовательности входных данных в последовательность выходных данных. В приложениях-собеседниках seq2seq может использоваться для генерации ответов на основе входных запросов.

7. BERT (Bidirectional Encoder Representations from Transformers): BERT представляет собой модель Transformer, обученную на огромном объеме текста. BERT обладает способностью понимать контекст запроса и является эффективным инструментом для создания приложений-собеседников с высокой точностью ответов.
8. GPT (Generative Pre-trained Transformer): GPT – это еще одна модель Transformer, предобученная на больших корпусах текста. GPT может использоваться в приложениях-собеседниках для генерации текста, включая ответы на запросы пользователей.

Использование комбинации вышеупомянутых типов нейронных сетей и моделей может значительно повысить качество приложений-собеседников, обеспечивая более точные и информативные ответы пользователям.

### **3.1 Рекуррентные нейронные сети**

Рекуррентные нейронные сети (RNN) представляют собой класс нейронных сетей, разработанных для обработки и анализа последовательностей данных. Они имеют способность сохранять информацию о предыдущих состояниях и использовать ее для обработки новых входных данных.

Рекуррентные нейронные сети были спроектированы для работы с последовательными данными, такими как тексты, речь или временные ряды. Главным принципом рекуррентных нейронных сетей является передача информации от предыдущего этапа обработки данных к текущему. Данный процесс обеспечивается за счет обратных связей, которые дают возможность модели сохранять информацию о предыдущих состояниях и использовать ее при анализе следующих элементов последовательности данных [1].

Рассмотрим особенности рекуррентных нейронных сетей:

1. Память о предыдущих состояниях: рекуррентные нейронные сети обладают внутренней памятью, которая позволяет им сохранять информацию о предыдущих состояниях, поэтому они являются подходящими для анализа данных, где порядок имеет значение, например, текста, аудио и временных рядов.

2. Обработка последовательностей: рекуррентные нейронные сети способны работать с последовательностями переменной длины, такими как предложения в тексте или временные отметки во временных рядах.
3. Обратное распространение градиентов: рекуррентные нейронные сети используют обратное распространение градиентов для обучения, что позволяет им адаптироваться к данным и улучшать свои предсказательные способности.

На рисунке 1 представлена рекуррентная нейронная сеть и её развернутое представление.

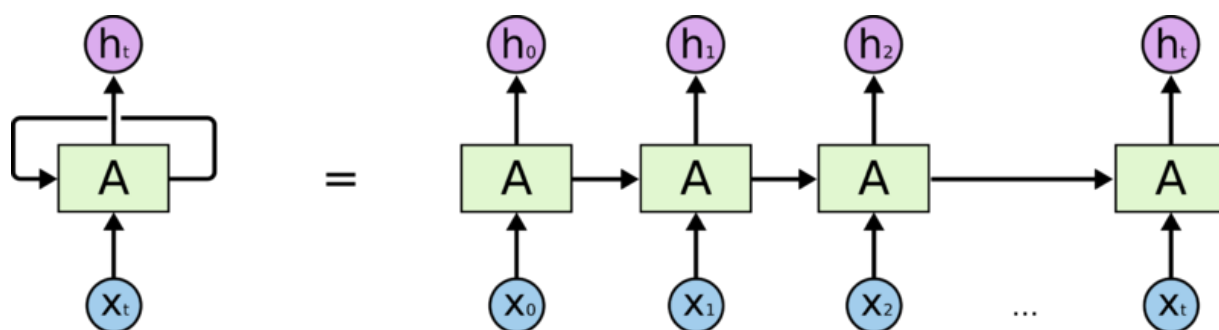


Рисунок 1 – RNN и её развернутое представление

Рассмотрим различные виды рекуррентных нейронных сетей [2].

Архитектура «один к одному» по сути является обычной нейронной сетью. На рисунке 2 изображена рекуррентная нейронная сеть «один к одному».

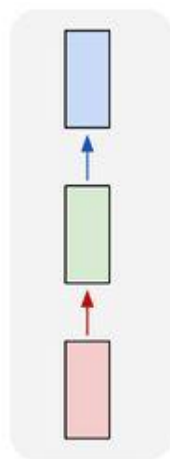


Рисунок 2 – RNN «один к одному»

Один вход ко многим выходам может применяться, например, для генерации аудиозаписи. На вход подается жанр музыки, на выходе получается последовательность аудиозаписи. На рисунке 3 изображена рекуррентная нейронная сеть «один ко многим».

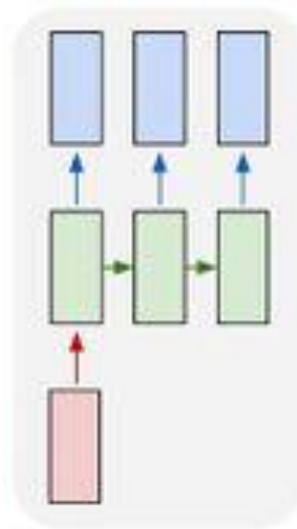


Рисунок 3 – RNN «один ко многим»

Много входов и один выход может применяться, например, для оценки тональности рецензии. На вход подаются слова рецензии, на выходе получается оценка ее тональности – позитивная рецензия или негативная. На рисунке 4 изображена рекуррентная нейронная сеть «многие к одному».

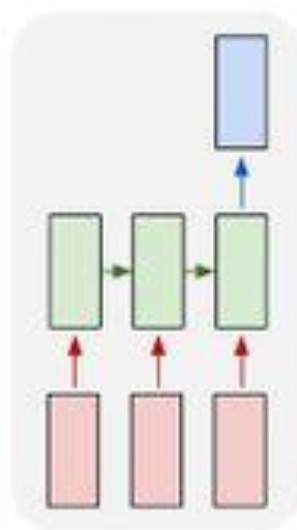


Рисунок 4 – RNN «многие к одному»

Архитектуру «многие ко многим» можно использовать для перевода текста с одного языка на другой. На рисунке 5 изображена рекуррентная нейронная сеть «многие ко многим».

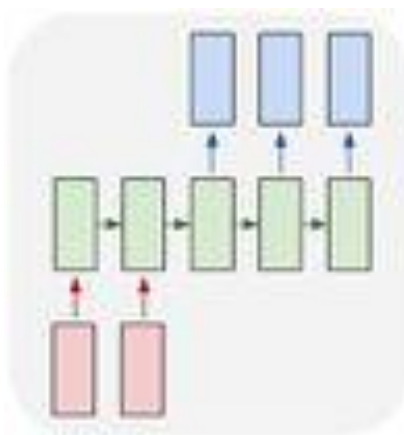


Рисунок 5 – RNN «многие ко многим»

Однако, рекуррентные нейронные сети обладают некоторыми ограничениями, такими как проблема затухающего градиента и неспособность эффективно учить долгосрочные зависимости. С целью преодоления данных проблем были разработаны более сложные архитектуры, такие как LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit), они имеют дополнительные механизмы для борьбы с этими ограничениями.

### **3.2 Долгая краткосрочная память**

Долгая краткосрочная память (Long short-term memory, LSTM) – особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучению долговременным зависимостям [3].

LSTM-модули разработаны специально для избежания проблемы долговременной зависимости. Они запоминают значения как на короткие, так и на длинные промежутки времени. Происходит это из-за того, что LSTM-модуль не использует функцию активации внутри своих рекуррентных компонентов, хранимое значение не размывается во времени, и градиент не исчезает при использовании метода обратного распространения ошибки.

Ключевые части LSTM-модуля – это состояние ячейки и различные фильтры. Состояние ячейки представляет из себя память сети, при помощи которой информация передается по всем модулям. Информация из ранних временных шагов может быть получена на более поздних.

В течение обучения состояние ячейки изменяется, информация добавляется или удаляется из состояния ячейки структурами, которые называются фильтрами. Фильтры контролируют поток информации на входах и на выходах модуля на основании некоторых условий, они состоят из слоя сигмоидальной нейронной сети и операции поточечного умножения.

Сигмоидальный слой возвращает числа в диапазоне от 0 до 1, они обозначают, какую долю каждого блока информации следует пропустить дальше по сети. Умножение на это значение используется для того, чтобы пропустить или запретить передачу потока информации внутрь и наружу памяти. Например, входной фильтр осуществляет контроль меры вхождения нового значения в память, а фильтр забывания осуществляет контроль меры сохранения значения в памяти. Выходной фильтр осуществляет контроль меры того, в какой степени значение, находящееся в памяти, используется при расчёте выходной функции активации [4].

На рисунке 6 представлена модель LSTM.

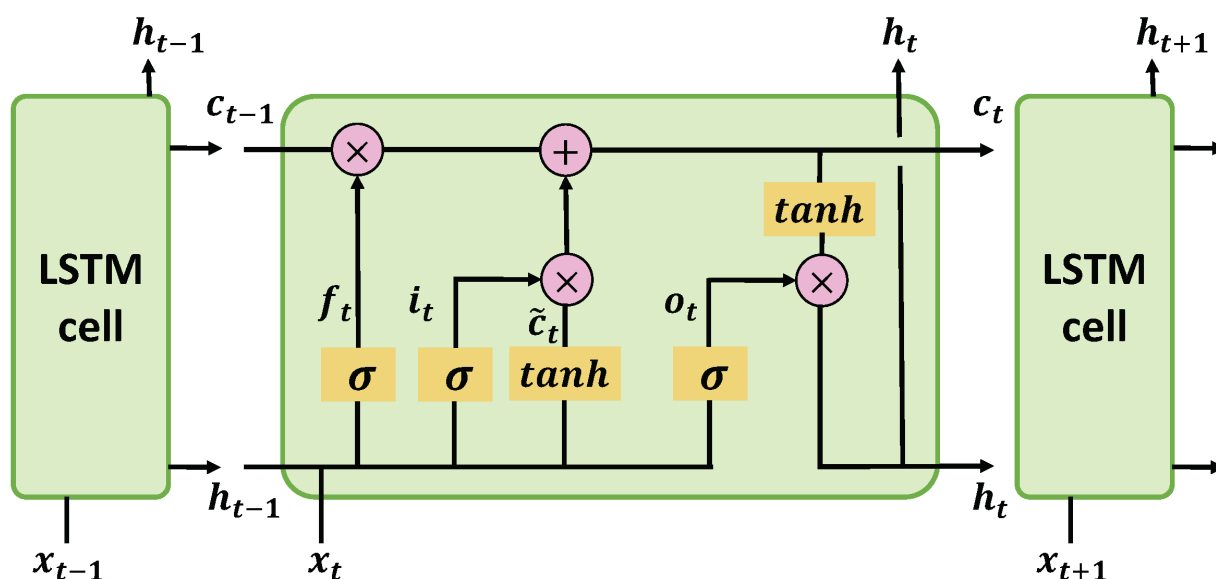
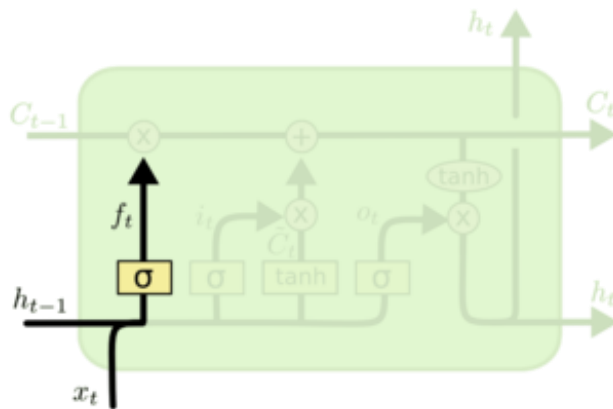


Рисунок 6 – Модель LSTM

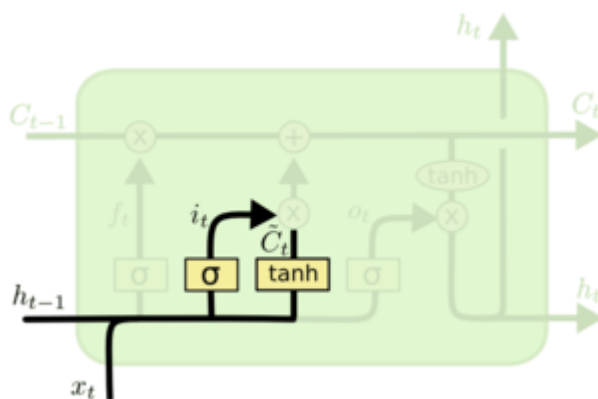
Рассмотрим принцип работы LSTM. На первом этапе «слой фильтра забывания» определяет, какую информацию можно забыть. Значения предыдущего выхода  $h_{t-1}$  и текущего входа  $x_t$  пропускаются через сигмоидальный слой. На рисунке 7 представлено вычисление  $f_t$ .  $W_f$  – матрица весов,  $b_f$  – вектор смещений.



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

Рисунок 7 – Вычисление «слоя фильтра забывания» в LSTM

Далее решается то, какая новая информация будет храниться в состоянии ячейки. Этот этап состоит из двух частей. Сначала сигмоидальный слой, «слой входного фильтра», определяет то, какие значения следует обновить. Затем tanh-слой строит вектор новых значений-кандидатов  $\tilde{C}_t$ , которые можно добавить в состояние ячейки. На рисунке 8 представлено вычисление  $i_t$  и  $\tilde{C}_t$ .  $W_i$ ,  $W_C$  – матрицы весов,  $b_i$ ,  $b_C$  – векторы смещений.

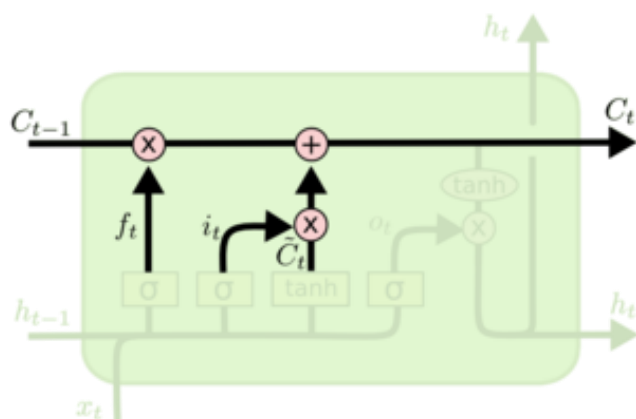


$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Рисунок 8 – Вычисление «слоя входного фильтра» и tanh-слоя в LSTM

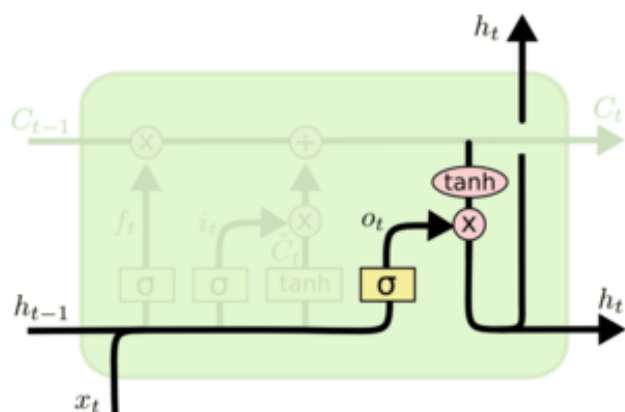
Чтобы заменить старое состояние ячейки  $C_{t-1}$  на новое состояние  $C_t$ , нужно умножить старое состояние на  $f_t$ . При этом забывается то, что решили забыть ранее. Далее прибавляется  $i_t * \tilde{C}_t$ . Это новые значения-кандидаты, умноженные на  $i_t$  – на сколько обновляется каждое из значений состояния. На рисунке 9 представлено вычисление  $C_t$ .



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Рисунок 9 – Вычисление нового состояния ячейки в LSTM

На последнем этапе определяется то, какая информация будет получена на выходе. Выходные данные основываются на нашем состоянии ячейки, к ним применяются некоторые фильтры. В начале значения предыдущего выхода  $h_{t-1}$  и текущего входа  $x_t$  пропускаются через сигмоидальный слой, он решает, какая информация из состояния ячейки будет выведена. Далее значения состояния ячейки проходят через tanh-слой и перемножаются с выходными значениями сигмоидального слоя, это позволяет выводить только требуемую информацию. На рисунке 10 представлено вычисление  $h_t$ .  $W_o$  – матрица весов,  $b_o$  – вектор смещений.



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

Рисунок 10 – Вычисление выходного значения ячейки в LSTM

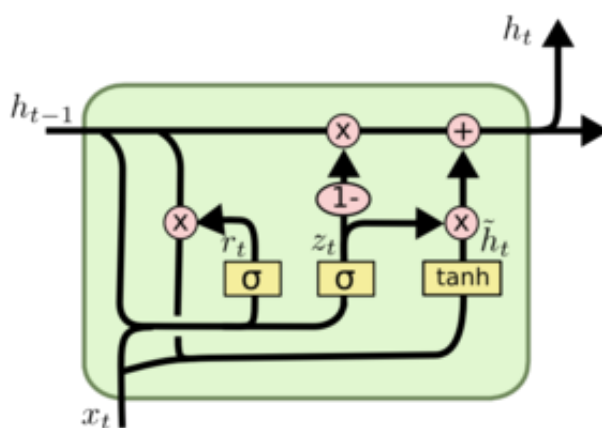


Полученные таким образом  $h_t$  и  $C_t$  передаются далее по цепочке.

### 3.3 Закрытый рекуррентный модуль

Закрытый рекуррентный модуль (Gated Recurrent Unit, GRU) является одним из видов рекуррентных нейронных сетей, аналогичных LSTM. Эта архитектура предназначена для упрощения и ускорения обучения по сравнению с LSTM. Она при этом сохраняет большую часть ее эффективности, особенно в задачах обработки последовательностей данных. Модель GRU основывается на тех же принципах, что и LSTM, но использует меньше фильтров и операций для вычисления нового скрытого состояния [5].

У GRU на один фильтр меньше, и фильтры несколько иначе соединены. Фильтры «забывания» и «входа» объединяются в один фильтр «обновления». Данный фильтр определяет, сколько информации сохраняется от последнего состояния и сколько информации получается от предыдущего слоя. Состояние ячейки объединяется со скрытым состоянием. Фильтр сброса состояния работает почти так же, как фильтр забывания, но расположен несколько иначе. Полная информация о состоянии отправляется на следующие слои. Выходной фильтр отсутствует. На рисунке 11 представлена модель GRU.  $W_z$ ,  $W_r$ ,  $W$  – матрицы весов.



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Рисунок 11 – Модель GRU

### 3.4 Механизм внимания

В основе механизма внимания лежит идея разделения операции оценки значимости вектора и получения информации из него. На вход механизма внимания поступает матрица размерности «длина текста» на «размер эмбединга». Эмбединг токена – это числовой вектор, содержащий информацию о признаках токена. В начале к вектору каждого слова независимо применяется некоторая нейронная сеть, имеющая один выход – оценку значимости соответствующего элемента текста, например, слова. Эта оценка является вещественным числом, которое для более значимых слов – больше, а для менее значимых слов – меньше. Термин «релевантность» применяется для обозначения оценки значимости слова, релевантность – это число, чем больше это число, тем слово более значимо. Часто релевантности слов рассчитываются с помощью однослойной нейронной сети, но может использоваться и более сложное преобразование. Релевантность для всех элементов текста рассчитывается с помощью одной и той же нейронной сети, веса которой не зависят от каждого конкретного слова. Далее оценки релевантности нормируются так, чтобы их сумма равнялась одному, для этого можно использовать функцию активации softmax. Затем оценки релевантности домножаются на соответствующие значения входных векторов и, таким образом, получается результирующий вектор. Размерность результирующего вектора такая же, как размерность эмбедингов слов на входе. Эту операцию можно записать через матричное произведение.

На рисунке 12 представлен пример механизма внимания. Words – матрица признаков токенов, Conv1d – однослойная нейронная сеть, имеющая один выход – оценку значимости соответствующего токена, Logits – вектор значений релевантностей токенов до применения softmax, Softmax – функция активации, Attention scores – вектор оценок релевантностей после применения функции softmax, сумма элементов данного вектора равняется единице, Result – результирующий вектор, его размерность такая же, как и размерность эмбедингов слов на входе.

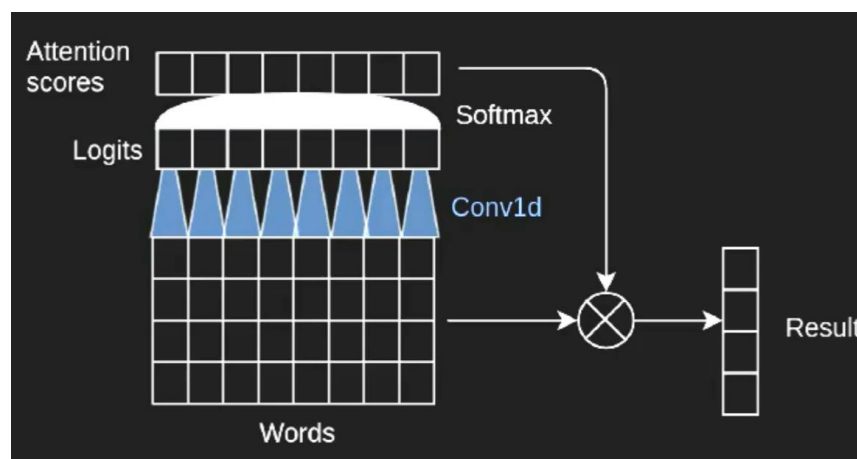


Рисунок 12 – Механизм внимания

Механизм внимания состоит из следующих основных компонентов. Во-первых, на вход может подаваться не только матрица слов, но и вектор-запрос, на основе которого будут рассчитываться релевантности слов. Второй компонент – механизм расчёта релевантности. Третий компонент – это механизм вычисления значений, то есть векторов, которые будут складываться с весами, полученными в результате расчёта значимости слов. Четвёртый компонент – это механизм получения единого вектора, то есть агрегация. Обычно оценки релевантности сначала нормируются с помощью функции активации softmax, а затем входные вектора домножаются на эти оценки и поэлементно складываются. В результате получается гораздо большая гибкость по сравнению с обычной агрегацией, значения выходного вектора более не являются независимыми, теперь одни элементы входных векторов работают на привлечение внимания, а другие на передачу полезной информации [6].

Существует множество вариантов механизма внимания. Рассмотрим некоторые из них. Для получения результирующего вектора можно использовать не исходные векторы, а векторы, преобразованные с помощью другой нейронной сети. При этом веса у нейронных сетей, отвечающих за получение значений и за оценку релевантности, отличаются, что даёт больше гибкости.

Пример такого механизма внимания представлен на рисунке 13. Words – матрица признаков токенов, Conv1d – однослойные нейронные сети, нижняя отвечает за получение значений, верхняя – за оценку релевантностей, их веса различаются, Values – матрица значений, полученная с помощью нейронной сети, данная матрица будет перемножаться с вектором оценок релевантности, Logits – вектор значений релевантностей токенов до применения softmax, Softmax – функция активации, Attention scores – вектор оценок релевантности после применения функции softmax, сумма элементов данного вектора равняется единице, Result – результирующий вектор, его размерность такая же, как и размерность эмбедингов слов на входе.

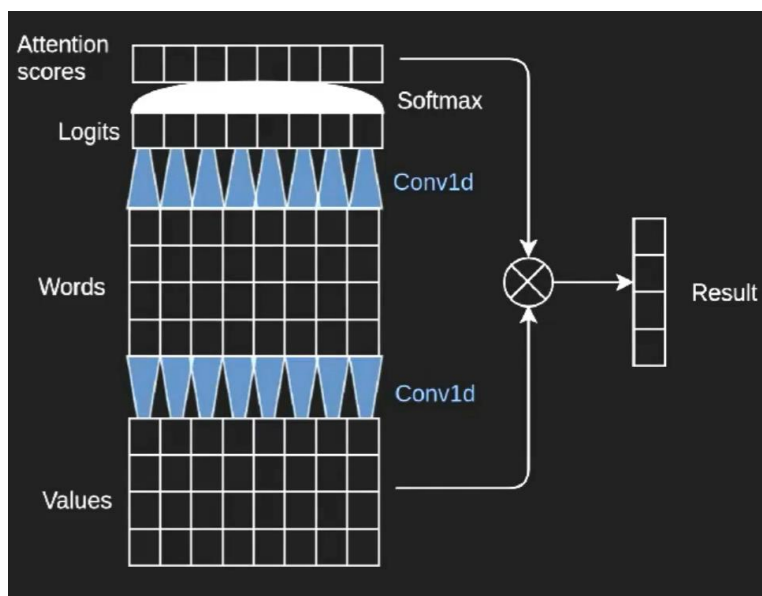


Рисунок 13 – Механизм внимания с преобразованием исходных векторов

Также задача может быть поставлена так, что слова во входной последовательности не являются релевантными или не релевантными сами по себе, их релевантность оценивают относительно запроса, то есть в контексте конкретной потребности пользователя. В таких случаях релевантность может оцениваться скалярным произведением вектора запроса и векторов из входной матрицы. Также это можно записать в виде матричного произведения. Чем ближе векторы входных слов к вектору запроса в смысле некоторой метрики, тем более они значимы.

На рисунке 14 приведен механизм внимания с запросом. Words – матрица признаков токенов, Query – вектор-запрос, Logits – вектор значений релевантностей токенов до применения softmax, Softmax – функция активации, Attention scores – вектор оценок релевантностей после применения функции softmax, сумма элементов данного вектора равняется единице, Result – результирующий вектор, его размерность такая же, как и размерность эмбеддингов слов на входе.

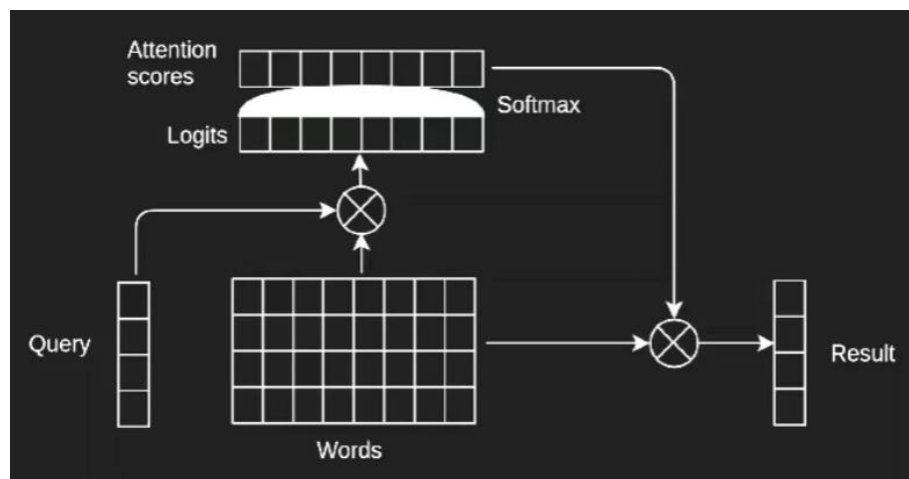


Рисунок 14 – Механизм внимания с запросом

Также можно не брать запрос извне, а вычислить самостоятельно – например, с помощью глобального пулинга с выбором максимума. Тогда оценка значимости каждого слова будет обусловлена на весь текст. Механизмы внимания, которые используют элементы одного и того же текста в качестве ключей и значений, называются механизмами «самовнимания» («self-attention»).

Рассмотрим механизм внутреннего внимания. В механизме внутреннего внимания на вход принимается матрица признаков токенов. Столбцы соответствуют токенам, а строки соответствуют признакам токенов. В результате применения внимания получается матрица, которая имеет такой же размер, что и входная матрица, но в ней уже каждый вектор содержит информацию о значении соответствующего токена в контексте всех остальных токенов. Алгоритм работы внутреннего внимания практически ничем не

отличается от обычного внимания, особенность заключается в том, что и в качестве запросов, и в качестве ключей, используются одни и те же данные, сами токены. В начале рассчитывается квадратная матрица попарного сходства каждого токена с каждым. Далее эта матрица, с помощью функции активации softmax, нормируется по строкам или по столбцам таким образом, чтобы сумма весов по строке или столбцу была равна одному. Затем исходные векторы выступают в роли значений. Они взвешиваются с помощью полученных весов и складываются, эту операцию можно записать с помощью матричного произведения.

Модель механизма внутреннего внимания представлена на рисунке 15. Words – матрица признаков токенов, Logits – квадратная матрица попарного сходства каждого токена с каждым, Softmax – функция активации, Scores – матрица, полученная из матрицы Logits нормированием при помощи функции softmax по строкам или по столбцам так, чтобы сумма весов по строке или столбцу равнялась единице, Result – результирующая матрица того же размера, что и входная матрица, в которой каждый вектор содержит информацию о значении соответствующего токена в контексте всех остальных токенов.

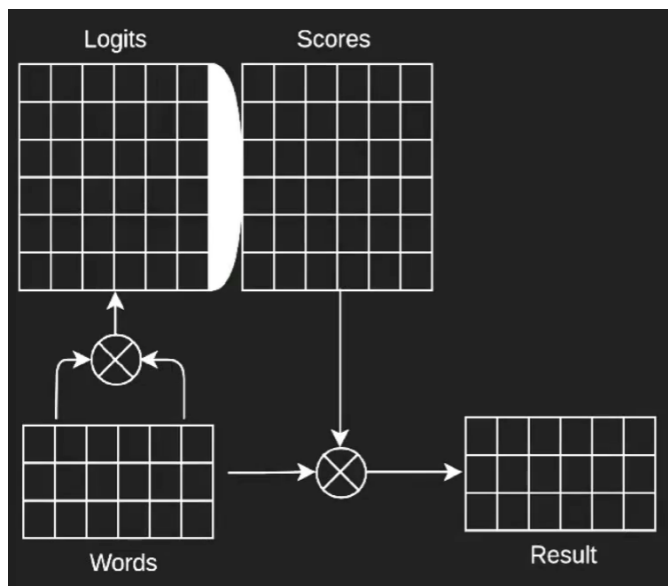


Рисунок 15 – Механизм внутреннего внимания

Рассмотренный вариант – это самый простой вариант внутреннего внимания, он показывает основной принцип работы. На практике обычно

используют более сложную схему. Отличие заключается в том, что перед тем, как считать попарное сходство токенов, исходные признаки токенов преобразуются, это делается двумя независимыми нейронными сетями. Часто используется простое линейное преобразование – домножение на матрицу. Матрицы проекции – это квадратные матрицы, и длина их стороны равна размеру эмбединга токена. Это позволяет получить намного больше гибкости. При расчёте релевантности токен будет иметь разные признаки, в зависимости от того, выступает он в качестве запроса или в качестве ключа.

Схема механизма внутреннего внимания с разделением запросов, ключей и значений представлена на рисунке 16. Words – матрица признаков токенов, Queries – матрица запросов, Keys – матрица ключей, Values – матрица значений, Logits – квадратная матрица попарного сходства каждого токена с каждым, Softmax – функция активации, Scores – матрица, полученная из матрицы Logits нормированием при помощи функции softmax по строкам или по столбцам так, чтобы сумма весов по строке или столбцу равнялась единице, Result – результирующая матрица того же размера, что и входная матрица, в которой каждый вектор содержит информацию о значении соответствующего токена в контексте всех остальных токенов.

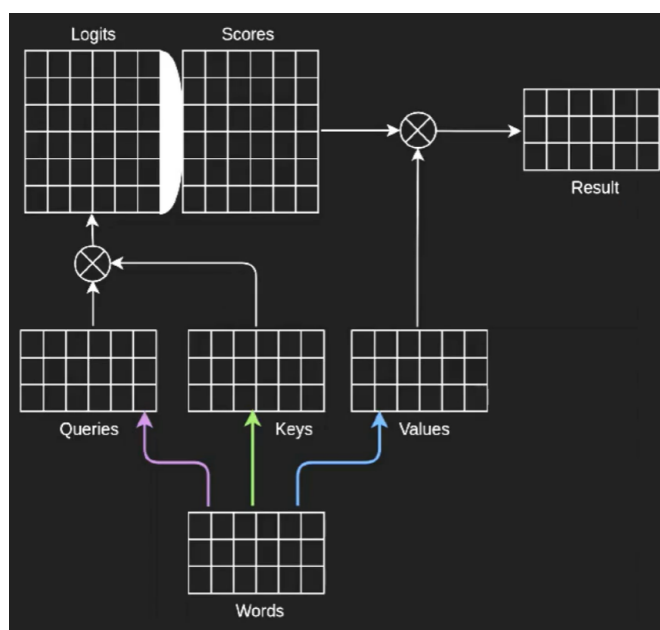


Рисунок 16 – Механизм внутреннего внимания с разделением запросов, ключей и значений

### 3.5 Нейронные сети трансформеры

Трансформер (Transformer) – архитектура глубоких нейронных сетей, основанная на механизме внимания. Основным преимуществом трансформеров по сравнению с рекуррентными нейронными сетями является их высокая эффективность в условиях параллелизма. Впервые модель трансформера была предложена в статье Attention is All You Need от разработчиков Google в 2017 году [6].

В трансформерах используется механизм внимания с несколькими «головами». «Головы» – это механизмы внутреннего внимания. Для получения проекций в разных «головах» используются разные веса, нормирование выполняется независимо в каждой «голове», что позволяет строить результирующий вектор признаков, учитывая множество аспектов, а не только один, как если бы использовался обычный механизм внутреннего внимания. Каждая «голова» работает с пространством признаков меньшего размера. Потом, на выходе, они конкатенируются. Размер выходной матрицы остаётся прежним – таким же, каким и был на входе.

На рисунке 17 представлен механизм внимания с несколькими «головами». Words – матрица признаков токенов, Queries – вектор запросов, Keys – вектор ключей, Values – вектор значений, Logits – квадратная матрица попарного сходства каждого токена с каждым, Softmax – функция активации, Scores – матрица, полученная из матрицы Logits нормированием при помощи функции softmax по строкам или по столбцам так, чтобы сумма весов по строке или столбцу равнялась единице, Multi-Head Self-Attention – механизм внутреннего внимания с несколькими «головами», Result – результирующая матрица.



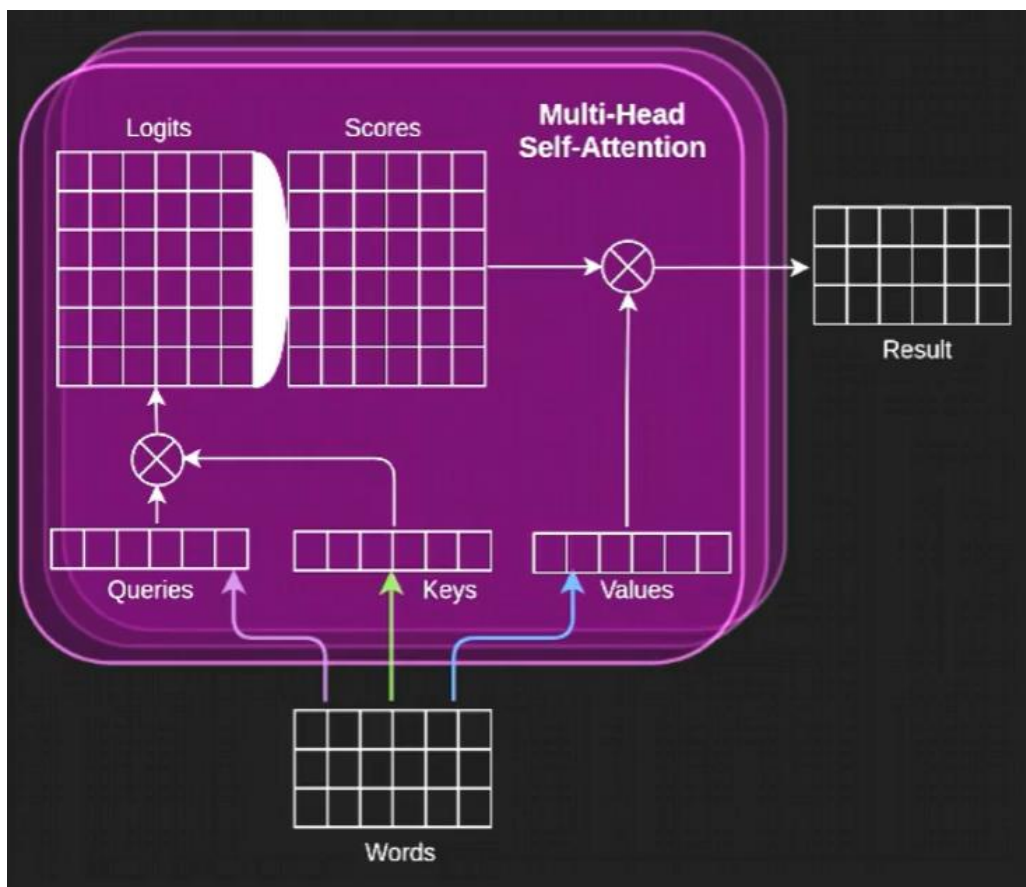


Рисунок 17 – Механизм внимания с несколькими «головами»

Ключевой идеей трансформера является внутреннее внимание с несколькими «головами». Но трансформер состоит не только из этих блоков. Он состоит из нескольких слоёв следующего вида: в начале используется внимание для учёта глобального контекста. Далее признаки каждого токена преобразовываются независимо с помощью двухслойной нейронной сети. Сеть применяется с одними и теми же параметрами ко всем токенам. Также есть связи в обход нелинейностей, что ускоряет процесс обучения за счёт лучшего протекания градиентов. Таких слоёв ставится несколько, один поверх другого [7]. Размерность данных эти слои не меняют.

На рисунке 18 представлена архитектура трансформера-кодировщика. Multi-Head Self-Attention – механизм внутреннего внимания с несколькими «головами», LayerNorm – применение функции активации softmax для нормирования значений.

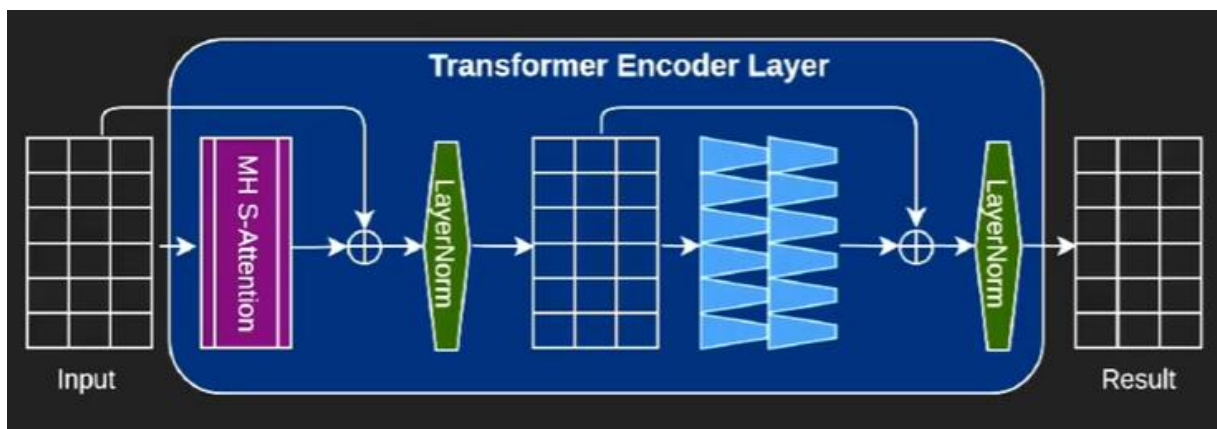


Рисунок 18 – Архитектура трансформера-кодировщика

На рисунке 19 представлена общая архитектура трансформера.

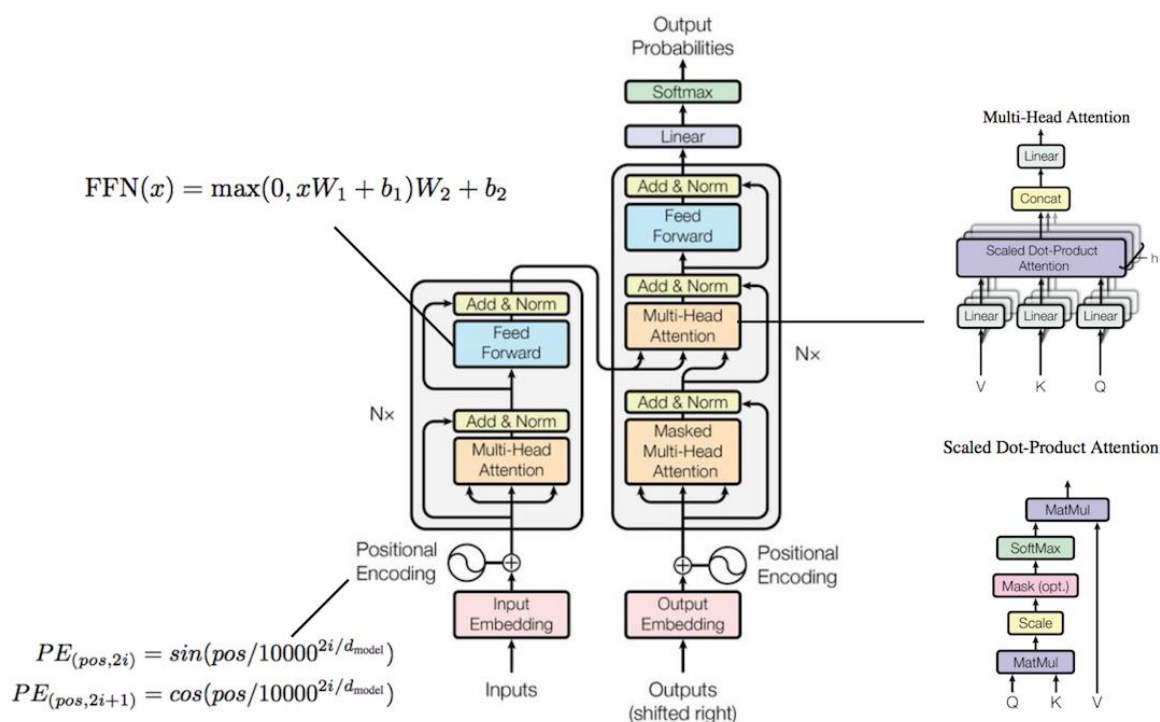


Figure 1: The Transformer - model architecture.

Рисунок 19 – Общая архитектура трансформера

Общая архитектура трансформера состоит из энкодера и декодера. Декодер отличается тем, что в нём используются маски. В него подаются выходы энкодера.

### 3.6 Модель «последовательность к последовательности»

Идея модели «последовательность к последовательности» (sequence to sequence) заключается в том, чтобы принимать последовательность переменной длины в качестве входных данных и возвращать последовательность переменной длины в качестве выходных данных, используя модель фиксированного размера.

Данную задачу можно выполнить, используя вместе две отдельные рекуррентные нейронные сети. Одна рекуррентная нейронная сеть действует как энкодер, который кодирует входную последовательность переменной длины в контекстный вектор фиксированной длины. Этот контекстный вектор (последний скрытый слой RNN) содержит семантическую информацию о предложении запроса. Вторая рекуррентная нейронная сеть – это декодер, который принимает входное слово и вектор контекста и возвращает предположение о следующем слове в последовательности и скрытое состояние для использования на следующей итерации [8]. На рисунке 20 изображен пример модели sequence to sequence.

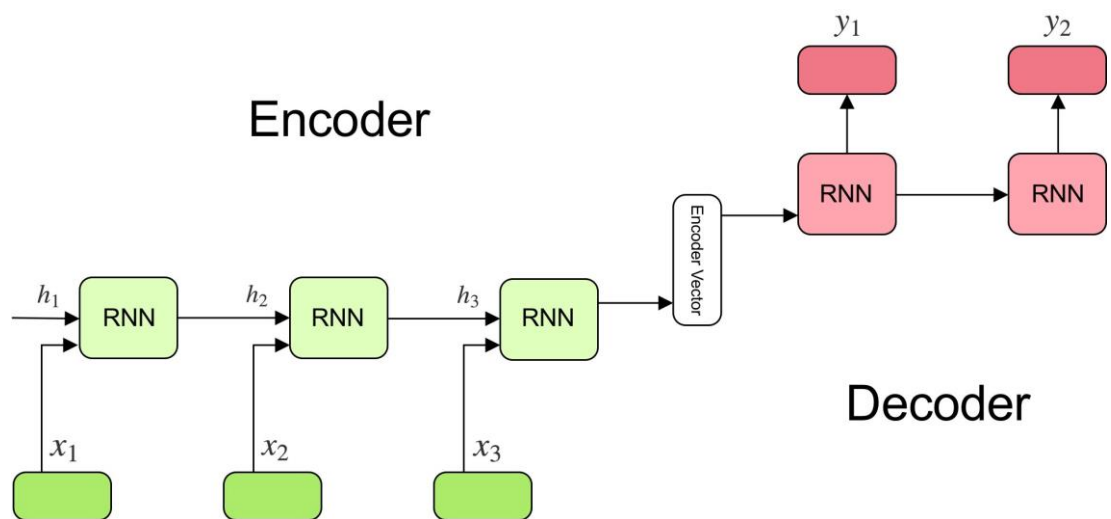


Рисунок 20 – Пример модели sequence to sequence

Энкодер RNN выполняет итерацию входного предложения по одному токenu, например, слову за раз, на каждом временном шаге выводя вектор «вывода» и вектор «скрытого состояния». Затем вектор «скрытого состояния»

передается на следующий временной шаг, выходной вектор записывается. Энкодер преобразует контекст, который он видел в каждой точке последовательности, в набор точек в многомерном пространстве, декодер будет использовать их для генерации вывода для данной задачи.

Декодер RNN генерирует ответное предложение поэтапно. Для генерации следующего слова в последовательности он использует контекстные векторы энкодера и внутренние скрытые состояния. Декодер RNN продолжает генерировать слова до тех пор, пока не выдаст EOS\_token, который представляет собой конец предложения. Распространенная проблема с декодером vanilla sequence to sequence, представляющим собой базовую модель декодера, заключается в том, что, если мы будем полагаться исключительно на контекстный вектор для кодирования значения всей входной последовательности, вполне вероятно, что мы потеряем информацию. Это особенно актуально при работе с длинными входными последовательностями, что значительно ограничивает возможности декодера.

Для борьбы с этим используется механизм внимания, который позволяет декодеру обращать внимание на определенные части входной последовательности, а не использовать весь фиксированный контекст на каждом шаге [9].

Для вычисления внимания используется текущее скрытое состояние декодера и выходные данные энкодера. Значения коэффициентов внимания на выходе имеют ту же форму, что и входная последовательность, что позволяет умножать их на выходные данные энкодера, получая взвешенную сумму, которая указывает, на какие части выходных данных энкодера следует обратить внимание [10].

### **3.7 Представления двунаправленного энкодера от трансформеров**

Представления двунаправленного энкодера от трансформеров (Bidirectional Encoder Representations from Transformers, BERT) – это мощная нейронная сеть, разработанная Google для обработки естественного языка. BERT построена на архитектуре трансформер, но с увеличенным числом и

размером слоев, отсутствующей декодирующей частью и глубокой двунаправленностью, то есть рассмотрением контекста с двух сторон [11].

Рассмотрим архитектуру BERT:

1. BERT основан на трансформере, который является моделью глубокого обучения для понимания контекста в тексте.
2. BERT использует двунаправленный подход, что означает, что он способен понимать контекст как слева направо, так и справа налево.

BERT имеет следующие преимущества:

1. Понимание контекста: BERT способен учитывать контекст предложения для более точного понимания значения слов.
2. Предварительное обучение: BERT предварительно обучается на больших объемах текста, что позволяет ему лучше понимать язык.
3. Универсальность: BERT может использоваться для различных задач NLP, таких как классификация, извлечение информации, вопросно-ответные системы и другие.

BERT находит применение в следующих областях:

1. Понимание текста: BERT может использоваться для понимания смысла текстов на естественном языке.
2. Автоматическая обработка языка: BERT используется для автоматической обработки языка, что полезно в поисковых системах, чат-ботах и других областях.

BERT представляет собой значительный прорыв в области NLP, благодаря своей способности понимать контекст и обрабатывать тексты на естественном языке более эффективно. Его универсальность и предварительное обучение делают его одним из самых мощных инструментов для работы с текстовыми данными в настоящее время.

### **3.8 Генеративный предварительно обученный трансформер**

Генеративный предварительно обученный трансформер (Generative Pre-trained Transformer, GPT) – это модель нейронной сети, разработанная компанией OpenAI для выполнения различных задач обработки естественного

языка. Она способна генерировать текст, отвечать на вопросы, завершать предложения и многое другое.

Оригинальный GPT и GPT-2 являются адаптациями Transformer – алгоритма 2017 года от Google. GPT-3 имеет ту же архитектуру, что и предыдущий алгоритм GPT-2. Главное отличие – количество параметров, используемых в модели, увеличили до 175 миллиардов. GPT-3 обучали на 570 гигабайтах текста или 1,5 триллионах слов [12].

GPT основана на технологии трансформеров, которая позволяет модели обрабатывать последовательности данных с учётом их контекста. GPT состоит из множества слоёв, каждый из которых обрабатывает информацию последовательно и передаёт её следующему слою для дальнейшей обработки.

GPT проходит через этап предварительного обучения на огромных объёмах текстовых данных, что позволяет ей улавливать широкий спектр языковых закономерностей. Это предварительное обучение позволяет модели в дальнейшем успешно выполнять различные задачи.

После предварительного обучения, GPT может быть дообучена под конкретную задачу, например, генерация текстов, ответы на вопросы в чат-ботах и многое другое.

GPT обладает способностью генерировать качественный и связный текст, соответствующий контексту, а также улавливать длинные зависимости в текстах, что делает её мощным инструментом для обработки естественного языка.

Нейронная сеть GPT представляет собой мощный инструмент для обработки естественного языка, способный генерировать высококачественные тексты и успешно выполнять разнообразные задачи в этой области.

#### **4 Выбор модели нейронной сети для реализации диалога с ботом**

Для реализации диалога с ботом была выбрана модель нейронной сети sequence to sequence, состоящая из энкодера RNN и декодера RNN, соединённых при помощи механизма внимания. Энкодер суммирует входную последовательность в набор векторов, а декодер воздействует на

закодированную входную последовательность с помощью механизма внимания и генерирует выходную последовательность по одному токenu. В качестве RNN как в энкодере, так и в декодере, были выбраны блоки GRU.

Рассмотрим преимущества выбранной модели:

1. Гибкость последовательной генерации. Эта модель позволяет генерировать выходную последовательность по одному токenu, обеспечивая гибкость в обработке входных и выходных данных.
2. Улучшенная способность перевода входной последовательности в выходную. Благодаря использованию механизма внимания в декодере, модель способна воздействовать на закодированную входную последовательность с учетом важности различных элементов, что улучшает способность модели к точному переводу.
3. Блоки GRU для эффективной обработки последовательностей. Выбор блоков GRU в качестве RNN как в энкодере, так и в декодере способствует эффективной обработке последовательностей и позволяет модели улавливать долгосрочные зависимости в данных.
4. Обучение на парных последовательностях. Модель sequence to sequence позволяет обучаться на парах входных и выходных последовательностей, что способствует обучению модели на задачах, таких как генерация текста или машинный перевод.

Эти преимущества делают модель sequence to sequence с энкодером RNN GRU, декодером RNN GRU и механизмом внимания эффективным инструментом для обработки последовательностей и решения различных задач обработки естественного языка, таких, как, например, создание приложений-собеседников.

## **5 Выбор стека технологий**

Для написания программы был выбран язык программирования Python и его библиотеки и модули:

- PyTorch – фреймворк глубокого обучения, он включает в себя набор

инструментов для работы с моделями [13];

- OpenCV – библиотека компьютерного зрения, которая предназначена для анализа, классификации и обработки изображений;
- face\_recognition – библиотека распознавания лиц [14];
- speech\_recognition – библиотека распознавания речи [15];
- psycopg2 – модуль для подключения к PostgreSQL, выполнения SQL-запросов и других операций с базой данных [16];
- os – модуль для работы с операционной системой;
- time – модуль для работы со временем;
- tkinter – библиотека для разработки графического интерфейса;
- PIL – библиотека Python Imaging Library для работы с изображениями;
- shutil – модуль для высокоуровневых операций с файлами и коллекциями файлов;
- pickle – модуль, который реализует двоичные протоколы для сериализации и десериализации структуры Python объекта;
- sys – модуль, который обеспечивает доступ к некоторым переменным и функциям, взаимодействующим с интерпретатором Python;
- re – модуль для работы с регулярными выражениями.

## **6 Реализация программы**

### **6.1 Реализация пользовательского интерфейса приложения**

Сначала был создан пользовательский интерфейс приложения. Для его создания использовалась библиотека tkinter.

В пользовательском интерфейсе приложения сверху отображается видеопоток с веб-камеры. Ниже находятся кнопки для переключения между интерфейсом для распознавания лиц и ввода имен и интерфейсом для общения с ботом.

#### **6.1.1 Реализация интерфейса распознавания лиц и ввода имен**



В интерфейсе для распознавания лиц и ввода имен находится строка для вывода сообщений от приложения, поле для ввода имени человека, находящегося перед камерой, кнопка для голосового ввода имени человека, находящегося перед камерой, поля для изменения имени – поля ввода текущего имени и нового имени, кнопка для очистки базы данных.

### 6.1.2 Реализация интерфейса общения с ботом

В интерфейсе для общения с ботом находится строка для вывода сообщений от приложения, пролистывающееся текстовое поле для отображения диалога с ботом, поле для ввода сообщения боту, кнопка для голосового ввода сообщения боту и кнопка для очистки диалога.

## 6.2 Реализация базы данных

### 6.2.1 Создание модели «сущность-связь», описание сущностей

Сначала была создана модель «сущность-связь». В данной модели базы данных содержится информация об изображениях, идентифицированных людях, не идентифицированных людях и диалогах.

На рисунке 21 представлена созданная модель «сущность – связь».

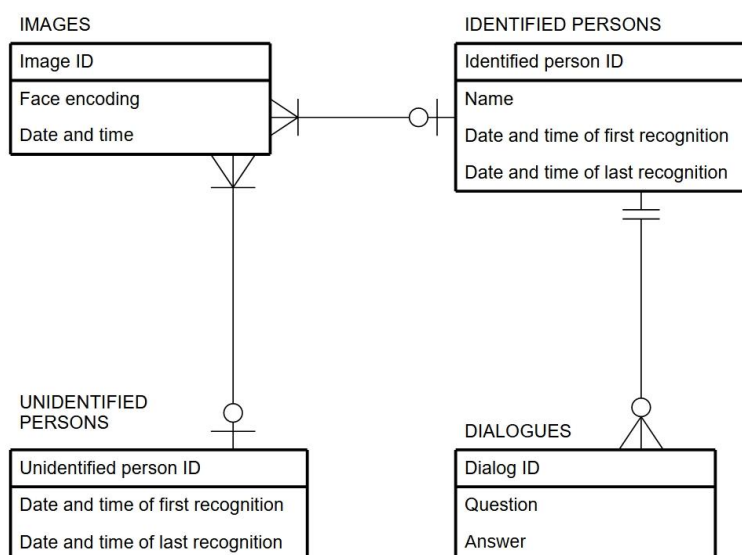


Рисунок 21 – Модель «сущность-связь»

Рассмотрим описание сущностей:

1. IMAGES – сущность, являющаяся абстракцией изображения.

Атрибуты:

- ImageId – идентификатор;
- FaceEncoding – кодировка лица;
- DateAndTime – дата и время сохранения изображения.

2. IDENTIFIED PERSONS – сущность, являющаяся абстракцией идентифицированного человека.

Атрибуты:

- IdentifiedPersonId – идентификатор;
- Name – имя;
- DateAndTimeOfFirstRecognition – дата и время первого распознавания камерой;
- DateAndTimeOfLastRecognition – дата и время последнего распознавания камерой.

3. UNIDENTIFIED PERSONS – сущность, являющаяся абстракцией неидентифицированного человека.

Атрибуты:

- UnidentifiedPersonId – идентификатор;
- DateAndTimeOfFirstRecognition – дата и время первого распознавания камерой;
- DateAndTimeOfLastRecognition – дата и время последнего распознавания камерой.

4. DIALOGUES – сущность, являющаяся абстракцией диалога идентифицированного человека с ботом.

Атрибуты:

- DialogId – идентификатор;
- Question – вопрос;
- Answer – ответ.

Рассмотрим связи между сущностями:

1. Рассмотрим связь между сущностями IMAGES и IDENTIFIED PERSONS. Изображению может соответствовать один идентифицированный человек, а может не соответствовать ни одного. У идентифицированного человека может быть 1 или более изображений.
2. Рассмотрим связь между сущностями IMAGES и UNIDENTIFIED PERSONS. Изображению может соответствовать один неидентифицированный человек, а может не соответствовать ни одного. У неидентифицированного человека может быть 1 или более изображений.
3. Рассмотрим связь между сущностями IDENTIFIED PERSONS и DIALOGUES. У идентифицированного человека может быть 0 или более диалогов. Диалогу соответствует ровно один идентифицированный человек.

### 6.2.2 Преобразование модели «сущность-связь» в реляционную модель

Далее модель «сущность-связь» была преобразована в реляционную модель. На рисунке 22 представлена реляционная модель базы данных.

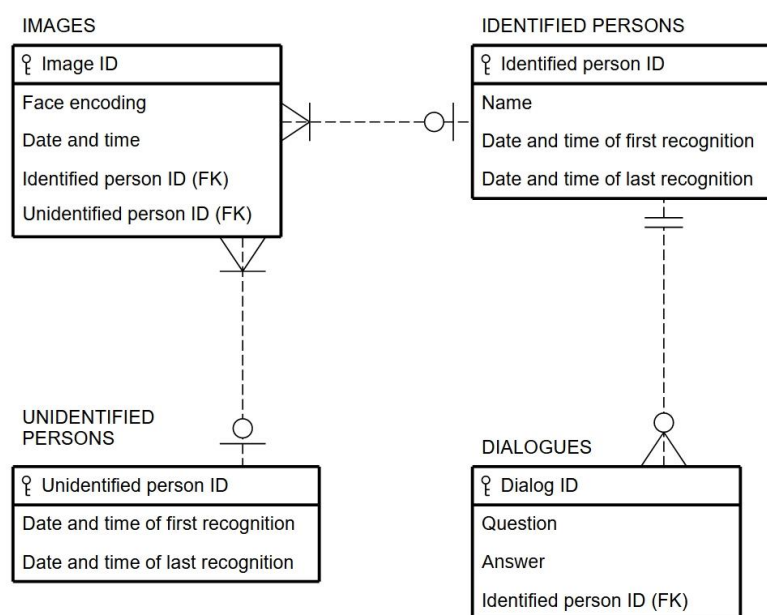


Рисунок 22 – Реляционная модель базы данных

### 6.2.3 Описание таблиц

Опишем таблицы реляционной модели.

В таблице 1 представлены описания столбцов для IMAGES.

Таблица 1 – IMAGES

Column name	Type	Key	NULL Status	Remarks
ImageId	SERIAL	PRIMARY KEY	NOT NULL	
FaceEncoding	BYTEA	No	NOT NULL	
DateAndTime	TIMESTAMP	No	NOT NULL	
IdentifiedPersonId	INTEGER	FOREIGN KEY	NULL	
UnidentifiedPersonId	INTEGER	FOREIGN KEY	NULL	

В таблице 2 представлены описания столбцов для IDENTIFIED PERSONS.

Таблица 2 – IDENTIFIED PERSONS

Column name	Type	Key	NULL Status	Remarks
IdentifiedPersonId	SERIAL	PRIMARY KEY	NOT NULL	
Name	varchar(50)	No	NOT NULL	UNIQUE
DateAndTimeOfFirstRecognition	TIMESTAMP	No	NOT NULL	
DateAndTimeOfLastRecognition	TIMESTAMP	No	NOT	

			NULL	
--	--	--	------	--

В таблице 3 представлены описания столбцов для UNIDENTIFIED PERSONS.

Таблица 3 – UNIDENTIFIED PERSONS

Column name	Type	Key	NULL Status	Remarks
UnidentifiedPersonId	SERIAL	PRIMARY KEY	NOT NULL	
DateAndTimeOfFirstRecognition	TIMESTAMP	No	NOT NULL	
DateAndTimeOfLastRecognition	TIMESTAMP	No	NOT NULL	

В таблице 4 представлены описания столбцов для DIALOGUES.

Таблица 4 – DIALOGUES

Column name	Type	Key	NULL Status	Remarks
DialogId	SERIAL	PRIMARY KEY	NOT NULL	
Question	varchar(150)	No	NULL	
Answer	varchar(150)	No	NULL	

### 6.3 Реализация распознавания лиц

Вверху интерфейса отображается видеопоток, получаемый с камеры компьютера. Распознавание и идентификация лиц осуществляется в функции face\_rec(). Сначала эта функция производит обнаружение лица в видеопотоке,

получаемом с камеры компьютера. При обнаружении лицо выделяется прямоугольником. Далее функция проходит по всем местоположениям и кодировкам лиц текущего кадра и сравнивает текущую кодировку лица с кодировками лиц на изображениях, сохраненных в таблицу IMAGES. Если кодировка лица совпадает с кодировкой лица изображения из таблицы IMAGES, то внизу прямоугольника пишется имя человека или, если человек не был идентифицирован, то строка «person», за которой следует id неидентифицированного человека. Если лиц с текущей кодировкой не было найдено в таблице IMAGES, то, значит, камера обнаружила нового человека. Информация о нём добавляется в таблицу UNIDENTIFIED PERSONS. Значениями DateAndTimeOfFirstRecognition и DateAndTimeOfLastRecognition становится текущий момент времени. Каждые 3 секунды информация об изображении сохраняется в таблицу IMAGES, а само изображение сохраняется в папке dataset в папку с именем человека, которому принадлежит эта фотография. На рисунках 23 – 24 изображена часть кода функции face\_rec().

```
def face_rec():
    global last_time, new_time, name_now, number_of_persons_in_front_of_the_camera, \
        names_persons_in_front_of_the_camera_now, names_persons_in_front_of_the_camera_last, interface2_label
    success, image = cap.read()
    locations = face_recognition.face_locations(image)
    encodings = face_recognition.face_encodings(image, locations)
    is_3_sec_passed = False
    number_of_persons_in_front_of_the_camera = len(locations)
    names_persons_in_front_of_the_camera_now = []
    for face_location, face_encoding in zip(locations, encodings):
        name = ""
        recognized_identified_person_id = 0
        recognized_unidentified_person_id = 0
        identified_person_ids = get_ids_from_table_identified_persons()
        unidentified_person_ids = get_ids_from_table_unidentified_persons()
        if identified_person_ids:
            identified_person_number = len(identified_person_ids)
        else:
            identified_person_number = 0
        if unidentified_person_ids:
            unidentified_person_number = len(unidentified_person_ids)
        else:
            unidentified_person_number = 0
        for i in range(identified_person_number):
            identified_person_id = identified_person_ids[i][0]
            identified_person_face_encoding = find_identified_person_face_encoding_in_table_images(identified_person_id)
            if face_recognition.compare_faces([face_encoding], identified_person_face_encoding)[0]:
                recognized_identified_person_id = identified_person_id
                name = get_name_by_id_in_table_identified_persons(identified_person_id)
                break
        for i in range(unidentified_person_number):
            unidentified_person_id = unidentified_person_ids[i][0]
            unidentified_person_face_encoding = find_unidentified_person_face_encoding_in_table_images(
                unidentified_person_id)
            if face_recognition.compare_faces([face_encoding], unidentified_person_face_encoding)[0]:
                recognized_unidentified_person_id = unidentified_person_id
                name = "person" + str(recognized_unidentified_person_id)
                break
```

Рисунок 23 – Код функции face\_rec()

```

top, right, bottom, left = face_location
left_top = (left, top)
right_bottom = (right, bottom)
color = [255, 0, 0]
cv2.rectangle(image, left_top, right_bottom, color, 4)
new_time = time.time()
if new_time - last_time >= 3:
    is_3_sec_passed = True
    face_img = image[top:bottom, left:right]
    face_img = cv2.cvtColor(face_img, cv2.COLOR_BGR2RGB)
    pil_img = Image.fromarray(face_img)
    if recognized_identified_person_id != 0:
        count = count_images_of_identified_person(recognized_identified_person_id) + 1
        pil_img.save("dataset/" + name + "/" + f"img_{count}.jpg")
        insert_identified_person_image_into_images(pickle.dumps(face_encoding), recognized_identified_person_id)
        update_date_and_time_of_last_recognition_in_table_identified_persons(recognized_identified_person_id)
    elif recognized_unidentified_person_id != 0:
        count = count_images_of_unidentified_person(recognized_unidentified_person_id) + 1
        pil_img.save("dataset/" + name + "/" + f"img_{count}.jpg")
        insert_unidentified_person_image_into_images(pickle.dumps(face_encoding),
                                                    recognized_unidentified_person_id)
        update_date_and_time_of_last_recognition_in_table_unidentified_persons(
            recognized_unidentified_person_id)
    else:
        count = identified_person_number + unidentified_person_number + 1
        name = f"person{count}"
        os.makedirs("dataset/" + name)
        pil_img.save("dataset/" + name + "/" + f"img_1.jpg")
        insert_into_unidentified_persons()
        insert_unidentified_person_image_into_images(pickle.dumps(face_encoding), count)
    left_bottom = (left, bottom)
    right_bottom = (right, bottom + 20)
    name_now = name
    names_persons_in_front_of_the_camera_now.append(name)
    cv2.rectangle(image, left_bottom, right_bottom, color, cv2.FILLED)
    cv2.putText(image, name, (left + 10, bottom + 15), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 4)
if is_3_sec_passed:
    last_time = new_time

```

Рисунок 24 – Код функции face\_rec()

## 6.4 Реализация чат-бота

### 6.4.1 Выбор данных для обучения

Для обучения нейронной сети были выбраны наборы данных диалогов из художественной литературы.

Для обучения нейронной сети был выбран набор из 1000000 пар предложений и ответов на них.

### 6.4.2 Предварительная обработка данных

В начале была проведена предварительная обработка данных. Диалоги были считаны из файлов, были сохранены пары вопросов и ответов. Предложения были приведены к нижнему регистру, были удалены знаки препинания, кроме основных. Были удалены диалоги, содержащие предложения, которые длиннее определенного количества символов. Затем был создан словарь для отображения слов в индексы. Были подсчитаны частоты

встречаемости каждого слова в диалогах, были удалены слишком редкие слова и диалоги, которые их содержат. Затем наборы данных для обучения были преобразованы в числовые тензоры.

### 6.4.3 Реализация выбранной модели нейронной сети

Для реализации чат-бота была выбрана модель нейронной сети sequence to sequence с энкодером RNN GRU, декодером RNN GRU и механизмом внимания. Данная модель обладает такими преимуществами, как гибкость последовательной генерации, улучшенная способность перевода входной последовательности в выходную, благодаря использованию механизма внимания, использование блоков GRU для эффективной обработки последовательностей, возможность обучения на парных последовательностях.

Энкодер RNN выполняет итерацию входного предложения по одному токену, например, слову за раз, на каждом временном шаге выводя вектор «вывода» и вектор «скрытого состояния». Затем вектор «скрытого состояния» передается на следующий временной шаг, выходной вектор записывается. Энкодер преобразует контекст, который он видел в каждой точке последовательности, в набор точек в многомерном пространстве, декодер будет использовать их для генерации вывода для данной задачи.

В данной работе был реализован энкодер, состоящий из двунаправленных GRU. Код класса, реализующего энкодер, представлен на рисунке 25.

```
class EncoderRNN(nn.Module):
    def __init__(self, hidden_size, embedding, n_layers=1, dropout=0):
        super(EncoderRNN, self).__init__()
        self.n_layers = n_layers
        self.hidden_size = hidden_size
        self.embedding = embedding
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers, dropout=(0 if n_layers == 1 else dropout),
                           bidirectional=True)

    def forward(self, input_seq, input_lengths, hidden=None):
        embedded = self.embedding(input_seq)
        packed = nn.utils.rnn.pack_padded_sequence(embedded, input_lengths)
        outputs, hidden = self.gru(packed, hidden)
        outputs, _ = nn.utils.rnn.pad_packed_sequence(outputs)
        outputs = outputs[:, :, :self.hidden_size] + outputs[:, :self.hidden_size:]
        return outputs, hidden
```

Рисунок 25 – Код класса, реализующего энкодер



Класс `EncoderRNN`, представляющий собой реализацию энкодера, является подклассом `nn.Module`. В методе `__init__` класса определяются основные параметры: `hidden_size` – размер скрытого состояния, `embedding` – векторные представления слов, `n_layers` – количество слоев GRU, `dropout` – коэффициент для дропаута. Затем инициализируется GRU слой с параметрами `hidden_size`, `n_layers`, `dropout` и флагом `bidirectional=True` для создания двунаправленного GRU. Метод `forward` определяет проход вперед через энкодер. Сначала векторы `input_seq` проходят через слой `embedding`. Затем они упаковываются в `padded sequence` с помощью `pack_padded_sequence` из `nn.utils.rnn`. Далее запускается слой GRU с упакованным входом и текущим скрытым состоянием. Результаты и скрытое состояние извлекаются из GRU, и далее производится операция по приведению размерности и объединению для получения окончательных выходных значений `outputs`. Метод возвращает `outputs` и последнее скрытое состояние.

Декодер RNN генерирует ответное предложение поэтапно. Он использует контекстные векторы энкодера и внутренние скрытые состояния для генерации следующего слова в последовательности. Слова продолжают генерироваться до тех пор, пока декодер не выдаст `EOS_token`, который представляет собой конец предложения. Распространенная проблема с базовой моделью декодера заключается в том, что, если мы будем полагаться исключительно на контекстный вектор для кодирования значения всей входной последовательности, вполне вероятно, что мы потеряем информацию.

Для борьбы с этим используется механизм внимания, который позволяет декодеру обращать внимание на определенные части входной последовательности, а не использовать весь фиксированный контекст на каждом шаге. Внимание вычисляется с использованием текущего скрытого состояния декодера и выходных данных энкодера. Веса внимания на выходе имеют ту же размерность, что и входная последовательность, что позволяет умножать их на выходные данные энкодера, получая взвешенную сумму,

которая указывает, на какие части выходных данных энкодера следует обратить внимание.

В данной работе была реализована усовершенствованная модель механизма внимания – «глобальное внимание». Основное отличие заключается в том, что при использовании «глобального внимания» учитываются все скрытые состояния энкодера, в отличие от «локального внимания», которое учитывает скрытое состояние энкодера только с текущего временного шага. Ещё одно отличие состоит в том, что при использовании «глобального внимания» рассчитываются весовые коэффициенты внимания с использованием скрытого состояния декодера только на текущем временном шаге.

Код класса, реализующего механизм внимания, представлен на рисунке 26.

```
class Attn(nn.Module):
    def __init__(self, hidden_size):
        super(Attn, self).__init__()
        self.hidden_size = hidden_size

    def dot_score(self, hidden, encoder_output):
        return torch.sum(hidden * encoder_output, dim=2)

    def forward(self, hidden, encoder_outputs):
        attn_energies = self.dot_score(hidden, encoder_outputs)
        attn_energies = attn_energies.t()
        return F.softmax(attn_energies, dim=1).unsqueeze(1)
```

Рисунок 26 – Код класса, реализующего механизм внимания

Класс Attn, реализующий механизм внимания, является подклассом nn.Module. В конструкторе этого класса осуществляется инициализация размера скрытого состояния. Метод dot\_score принимает на вход скрытое состояние и вывод энкодера, вычисляет скалярное произведение между ними с помощью функции torch.sum. В методе forward происходит вычисление энергий внимания путем применения dot\_score к скрытому состоянию и выводу

энкодера, затем эти значения транспонируются, и к ним применяется функция softmax для получения весов внимания.

На рисунке 27 представлен код класса, реализующего декодер.

```
class DecoderRNN(nn.Module):
    def __init__(self, embedding, hidden_size, output_size, n_layers=1, dropout=0.1):
        super(DecoderRNN, self).__init__()

        self.hidden_size = hidden_size
        self.output_size = output_size
        self.n_layers = n_layers
        self.dropout = dropout

        self.embedding = embedding
        self.embedding_dropout = nn.Dropout(dropout)
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers, dropout=(0 if n_layers == 1 else dropout))
        self.concat = nn.Linear(hidden_size * 2, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

        self.attn = Attn(hidden_size)

    def forward(self, input_step, last_hidden, encoder_outputs):
        embedded = self.embedding(input_step)
        embedded = self.embedding_dropout(embedded)
        rnn_output, hidden = self.gru(embedded, last_hidden)
        attn_weights = self.attn(rnn_output, encoder_outputs)
        context = attn_weights.bmm(encoder_outputs.transpose(0, 1))
        rnn_output = rnn_output.squeeze(0)
        context = context.squeeze(1)
        concat_input = torch.cat((rnn_output, context), 1)
        concat_output = torch.tanh(self.concat(concat_input))
        output = self.out(concat_output)
        output = F.softmax(output, dim=1)
        return output, hidden
```

Рисунок 27 – Код класса, реализующего декодер

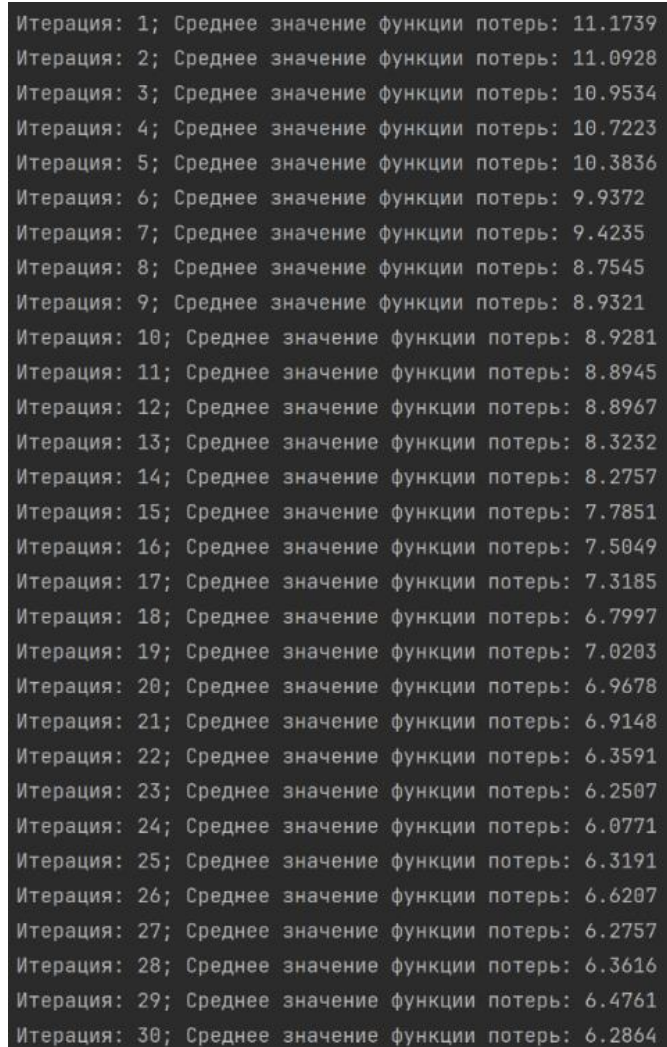
Класс DecoderRNN, реализующий декодер нейронной сети, является подклассом nn.Module. В конструкторе \_\_init\_\_ определены параметры, необходимые для построения декодера RNN, такие, как размерность скрытого слоя, размерность выхода, количество слоев и коэффициент дропаута. В методе forward определен прямой проход через декодер RNN. Сначала вход передается через слой эмбединга, после чего применяется дропаут. Затем вход передается через слой GRU, и полученные результаты применяются к механизму внимания для получения весов внимания. Далее выполняется конкатенация полученных величин. И в конце применяется функция softmax для получения итогового вывода.

## 7 Результаты работы

### 7.1 Результаты обучения нейронной сети

Для обучения нейронной сети был выполнен цикл из 100000 итераций, были посчитаны значения функции потерь на каждой итерации.

На рисунке 28 изображены значения функции потерь на первых 30 итерациях.



Итерация: 1;	Среднее значение функции потерь:	11.1739
Итерация: 2;	Среднее значение функции потерь:	11.0928
Итерация: 3;	Среднее значение функции потерь:	10.9534
Итерация: 4;	Среднее значение функции потерь:	10.7223
Итерация: 5;	Среднее значение функции потерь:	10.3836
Итерация: 6;	Среднее значение функции потерь:	9.9372
Итерация: 7;	Среднее значение функции потерь:	9.4235
Итерация: 8;	Среднее значение функции потерь:	8.7545
Итерация: 9;	Среднее значение функции потерь:	8.9321
Итерация: 10;	Среднее значение функции потерь:	8.9281
Итерация: 11;	Среднее значение функции потерь:	8.8945
Итерация: 12;	Среднее значение функции потерь:	8.8967
Итерация: 13;	Среднее значение функции потерь:	8.3232
Итерация: 14;	Среднее значение функции потерь:	8.2757
Итерация: 15;	Среднее значение функции потерь:	7.7851
Итерация: 16;	Среднее значение функции потерь:	7.5049
Итерация: 17;	Среднее значение функции потерь:	7.3185
Итерация: 18;	Среднее значение функции потерь:	6.7997
Итерация: 19;	Среднее значение функции потерь:	7.0203
Итерация: 20;	Среднее значение функции потерь:	6.9678
Итерация: 21;	Среднее значение функции потерь:	6.9148
Итерация: 22;	Среднее значение функции потерь:	6.3591
Итерация: 23;	Среднее значение функции потерь:	6.2507
Итерация: 24;	Среднее значение функции потерь:	6.0771
Итерация: 25;	Среднее значение функции потерь:	6.3191
Итерация: 26;	Среднее значение функции потерь:	6.6207
Итерация: 27;	Среднее значение функции потерь:	6.2757
Итерация: 28;	Среднее значение функции потерь:	6.3616
Итерация: 29;	Среднее значение функции потерь:	6.4761
Итерация: 30;	Среднее значение функции потерь:	6.2864

Рисунок 28 – Значения функции потерь на первых 30 итерациях

На рисунке 29 изображены значения функции потерь на последних 30 итерациях.

Итерация: 99971; Среднее значение функции потерь: 3.2933
Итерация: 99972; Среднее значение функции потерь: 3.3935
Итерация: 99973; Среднее значение функции потерь: 3.6950
Итерация: 99974; Среднее значение функции потерь: 3.4904
Итерация: 99975; Среднее значение функции потерь: 3.2125
Итерация: 99976; Среднее значение функции потерь: 3.7007
Итерация: 99977; Среднее значение функции потерь: 3.1472
Итерация: 99978; Среднее значение функции потерь: 3.2306
Итерация: 99979; Среднее значение функции потерь: 2.9305
Итерация: 99980; Среднее значение функции потерь: 3.5073
Итерация: 99981; Среднее значение функции потерь: 3.2650
Итерация: 99982; Среднее значение функции потерь: 3.6861
Итерация: 99983; Среднее значение функции потерь: 3.2249
Итерация: 99984; Среднее значение функции потерь: 3.0601
Итерация: 99985; Среднее значение функции потерь: 3.2131
Итерация: 99986; Среднее значение функции потерь: 3.3172
Итерация: 99987; Среднее значение функции потерь: 3.3658
Итерация: 99988; Среднее значение функции потерь: 3.0868
Итерация: 99989; Среднее значение функции потерь: 3.6707
Итерация: 99990; Среднее значение функции потерь: 3.0057
Итерация: 99991; Среднее значение функции потерь: 3.2834
Итерация: 99992; Среднее значение функции потерь: 3.4867
Итерация: 99993; Среднее значение функции потерь: 3.2626
Итерация: 99994; Среднее значение функции потерь: 3.1734
Итерация: 99995; Среднее значение функции потерь: 3.2017
Итерация: 99996; Среднее значение функции потерь: 3.5098
Итерация: 99997; Среднее значение функции потерь: 3.2328
Итерация: 99998; Среднее значение функции потерь: 3.4507
Итерация: 99999; Среднее значение функции потерь: 3.1465
Итерация: 100000; Среднее значение функции потерь: 3.3539

Рисунок 29 – Значения функции потерь на последних 30 итерациях

## 7.2 Пользовательский интерфейс приложения

В пользовательском интерфейсе приложения сверху отображается видеопоток с веб-камеры. Ниже находятся кнопки для переключения между интерфейсом для распознавания лиц и ввода имен и интерфейсом для общения с ботом.

В интерфейсе для распознавания лиц и ввода имен находится строка для вывода сообщений от приложения, поле для ввода имени человека, находящегося перед камерой, кнопка для голосового ввода имени человека, находящегося перед камерой, поля для изменения имени – поля ввода текущего имени и нового имени, кнопка для очистки базы данных. На рисунке 30 изображен интерфейс для распознавания лиц и ввода имен.

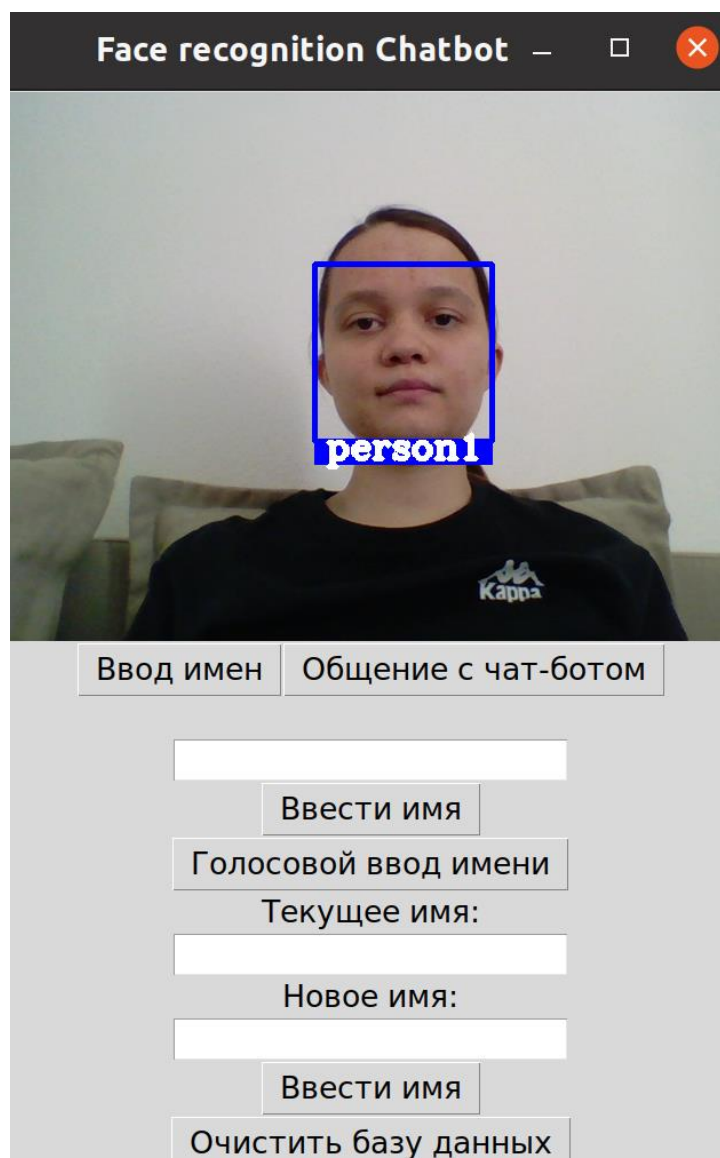


Рисунок 30 – Интерфейс для распознавания лиц и ввода имен

В интерфейсе для общения с ботом находится строка для вывода сообщений от приложения, пролистывающееся текстовое поле для отображения диалога с ботом, поле для ввода сообщения боту, кнопка для голосового ввода сообщения боту и кнопка для очистки диалога. На рисунке 31 изображен интерфейс для общения с ботом.

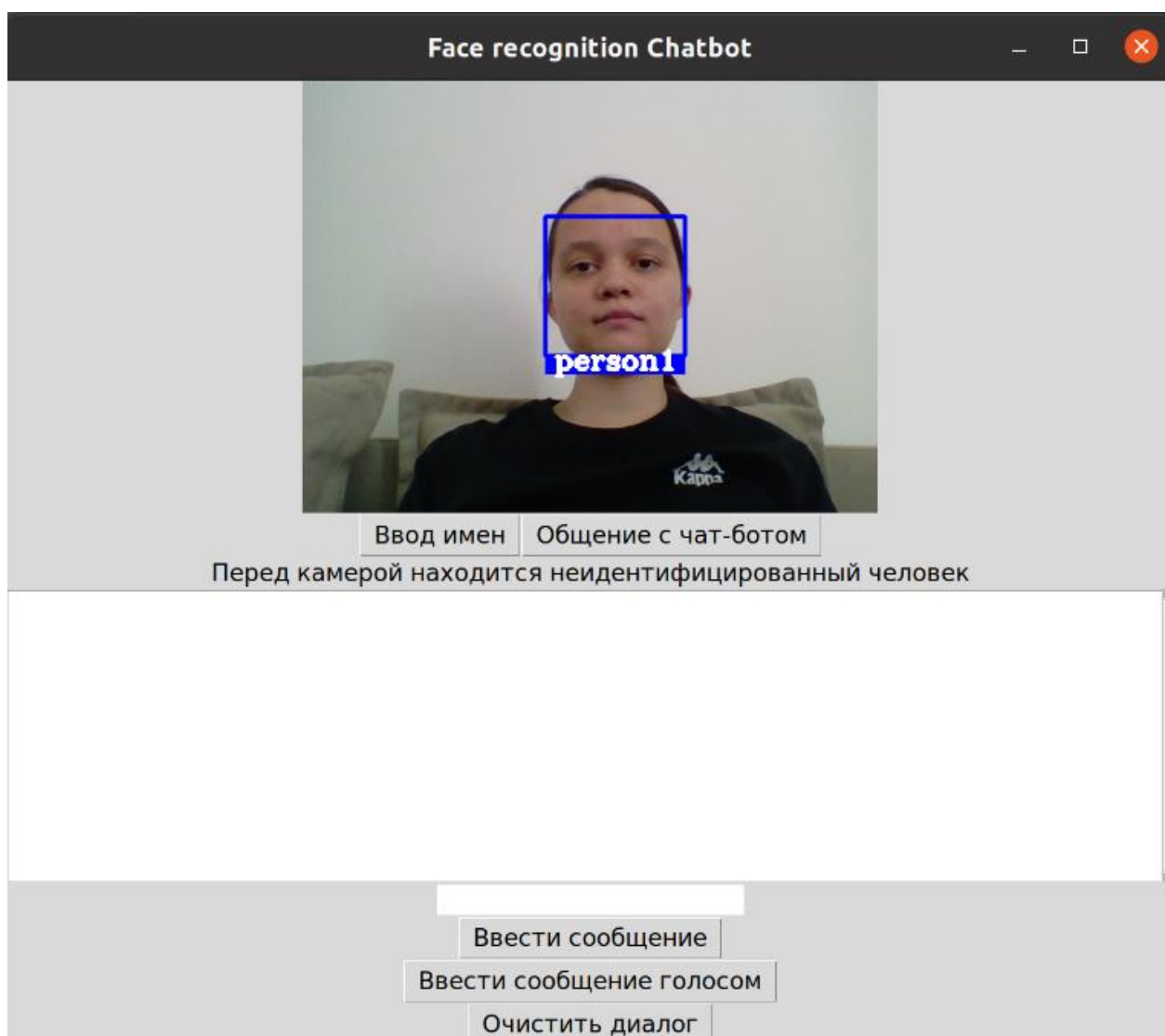


Рисунок 31 – Интерфейс для общения с ботом

### 7.3 Распознавание лиц и ввод имен

В приложении отображается видеопоток с веб-камеры. Происходит распознавание лиц людей, лица выделяются прямоугольниками, внизу подписывается имя человека, если оно было введено, или «person» и индекс человека, находящегося перед камерой, если имя не было введено. В интерфейсе распознавания лиц и ввода имен имена можно вводить текстом или голосом. На рисунках 32 – 33 изображен ввод имен людей, распознанных камерой.



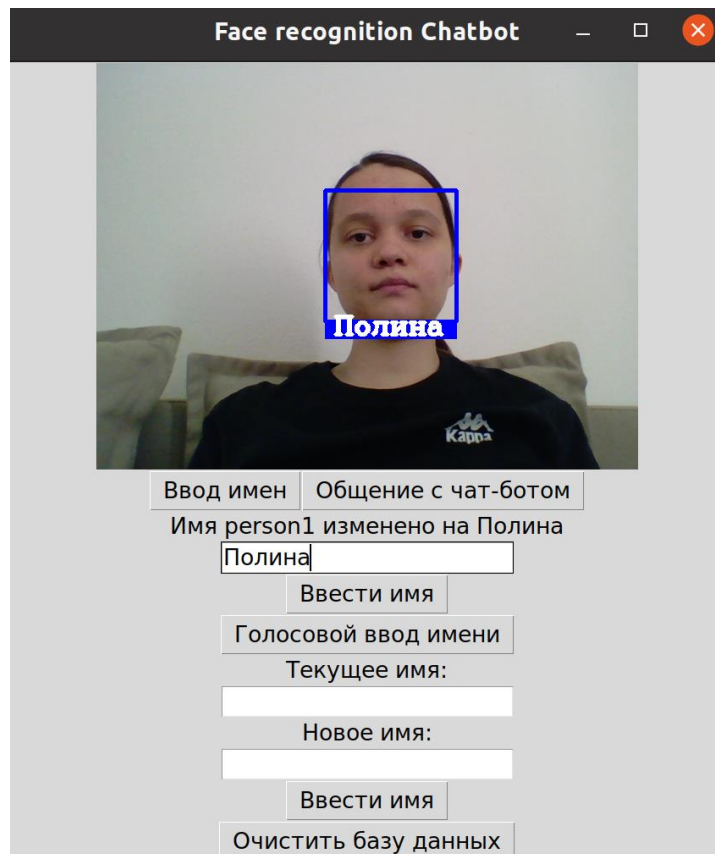


Рисунок 32 – Ввод имени человека, распознанного камерой

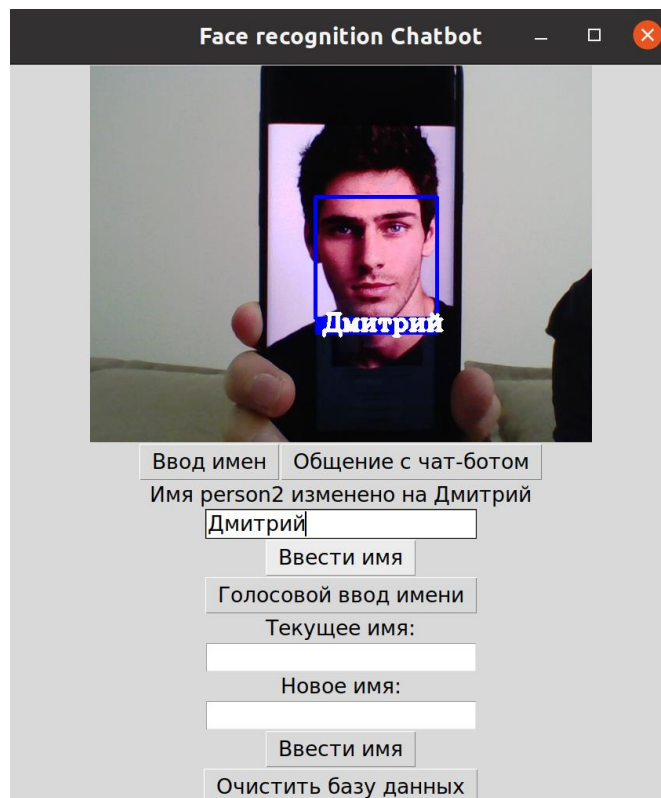


Рисунок 33 – Ввод имени человека, распознанного камерой



Изображения лиц разных людей сохраняются в разные папки. Сохранение фотографий разных людей в разные папки изображено на рисунках 34 – 36.

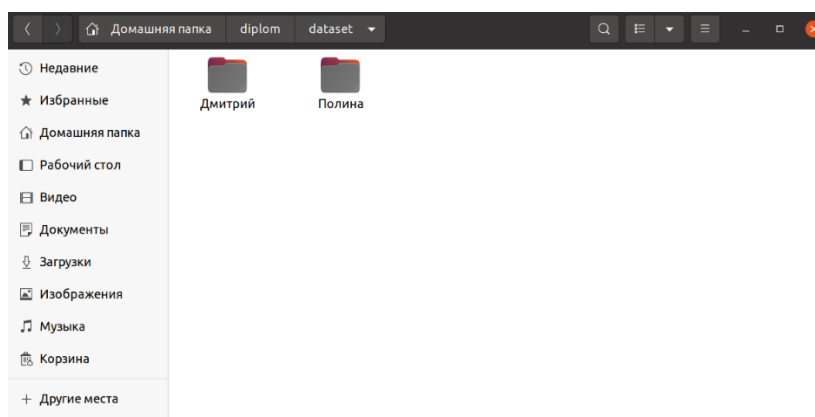


Рисунок 34 – Сохранение изображений лиц разных людей в разные папки

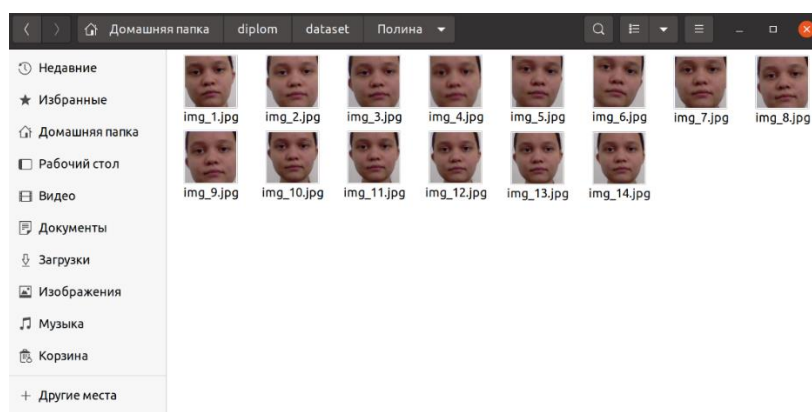


Рисунок 35 – Сохранение изображений лиц разных людей в разные папки

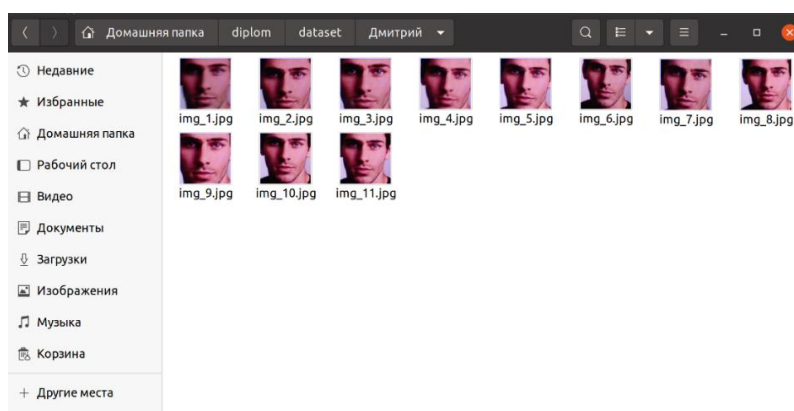


Рисунок 36 – Сохранение изображений лиц разных людей в разные папки

## 7.4 Тестирование проведения диалога с ботом

Когда человек подходит к камере, в интерфейсе общения с ботом отображается сохраненный диалог этого человека с ботом. Также можно вводить новые сообщения текстом или голосом.

На рисунках 37 и 38 изображены примеры диалогов двух людей с ботом. Когда человек подходит к камере, отображается сохраненный диалог этого человека с ботом.

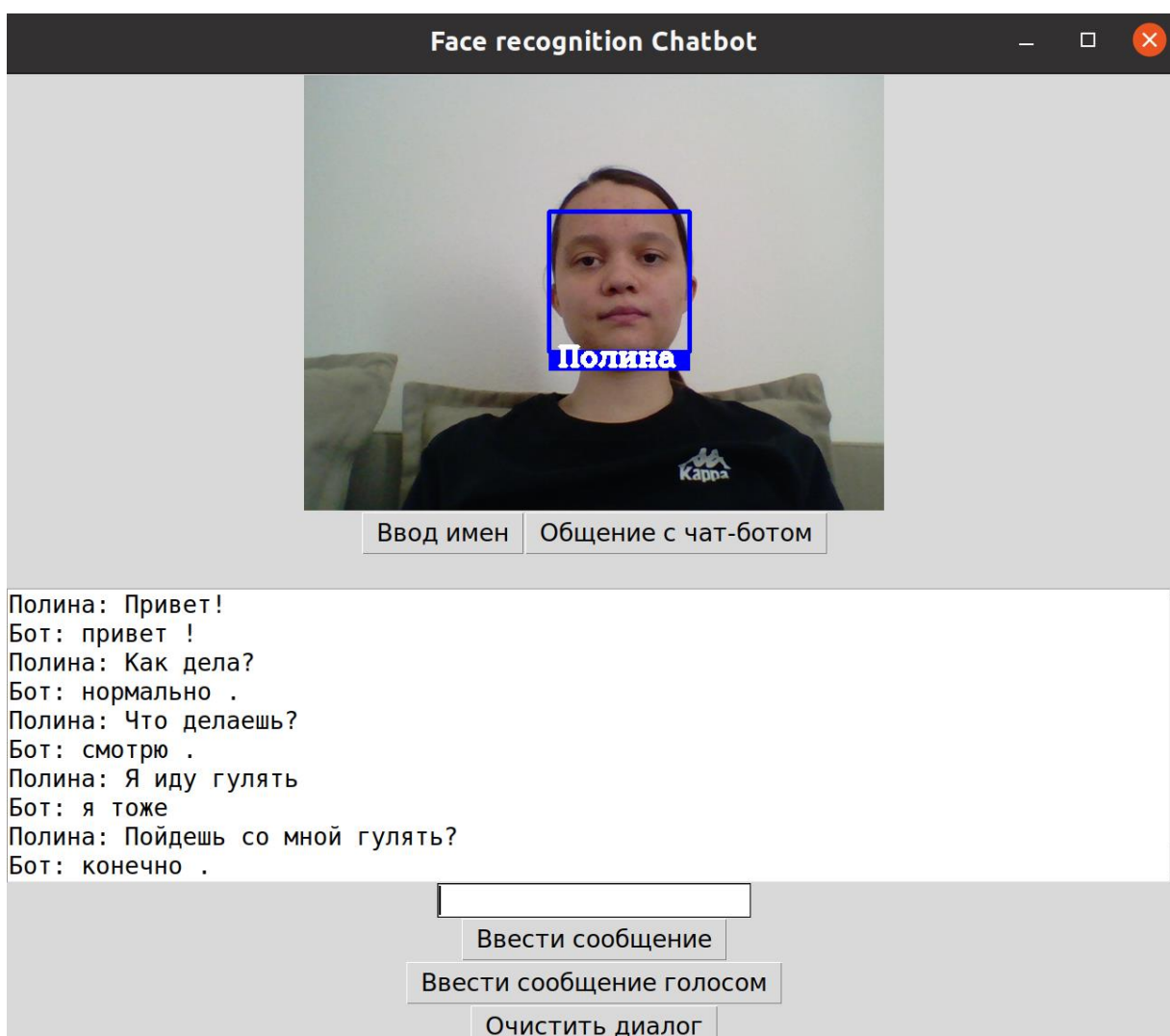


Рисунок 37 – Пример диалога человека с ботом

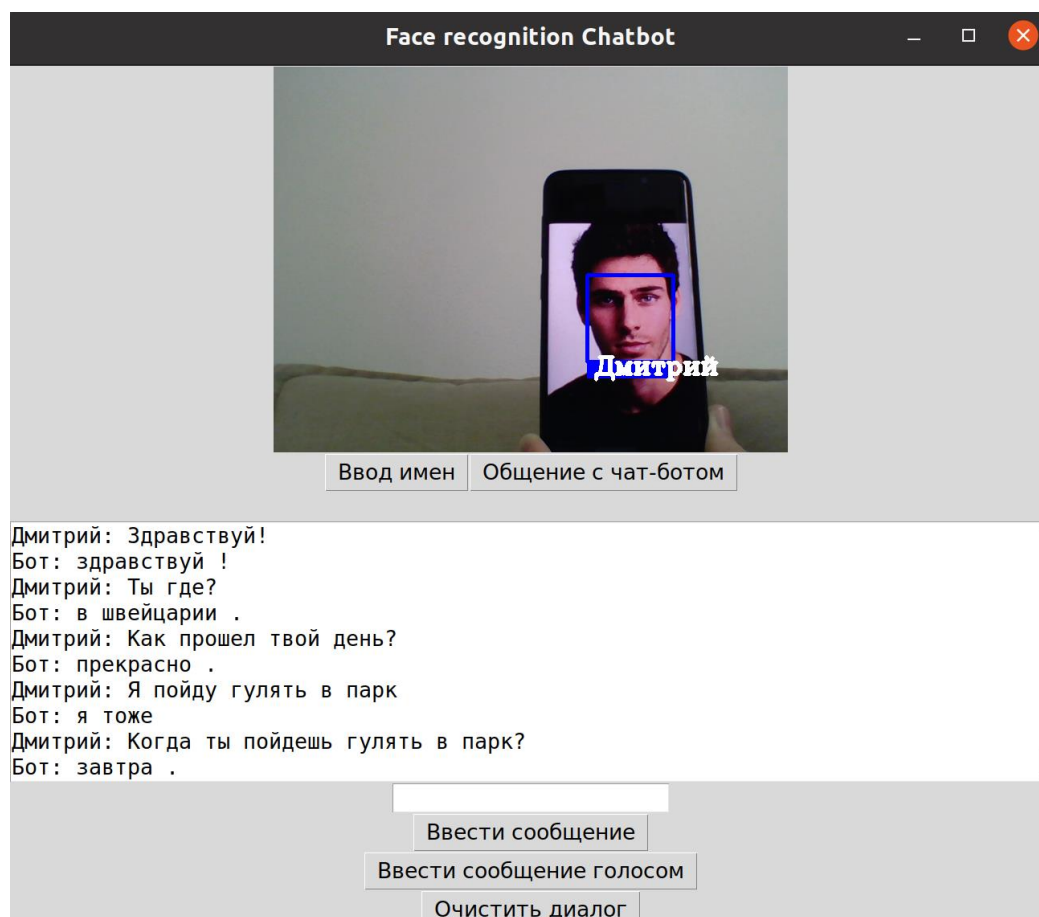


Рисунок 38 – Пример диалога человека с ботом

Если перед камерой никого нет, то отображается сообщение о том, что перед камерой нет людей, и диалоговое поле остается пустым. Такой пример изображен на рисунке 39.

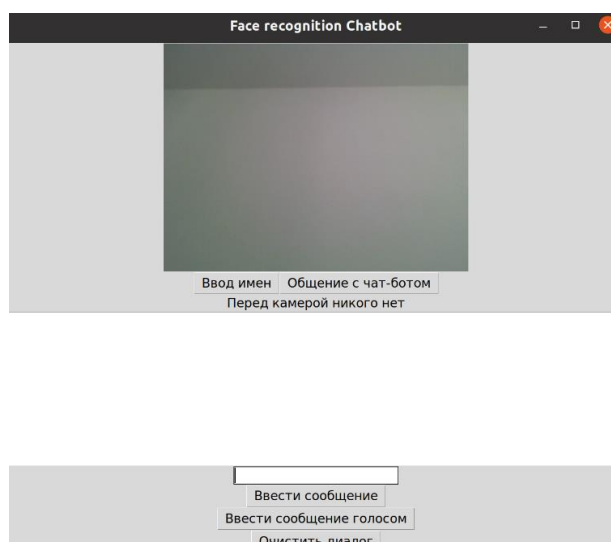


Рисунок 39 – Интерфейс общения с ботом, когда перед камерой никого нет

Если перед камерой находится более одного человека, отображается сообщение об этом, и диалоговое поле остается пустым. Такой пример изображен на рисунке 40.

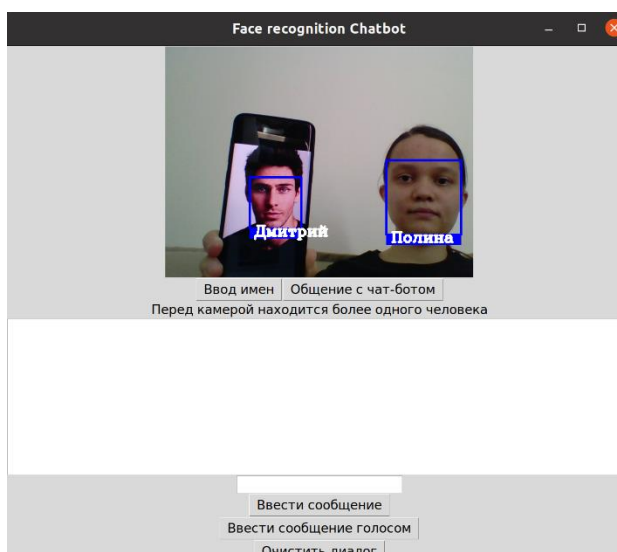


Рисунок 40 – Интерфейс общения с ботом, когда перед камерой находится более одного человека

Если перед камерой находится неидентифицированный человек, то есть человек, которому ранее не было введено имя, то отображается сообщение об этом, и диалоговое поле остается пустым. Такой пример изображен на рисунке 41.

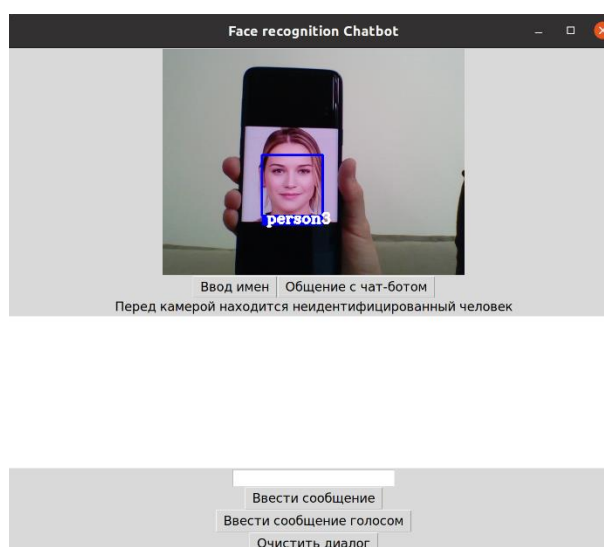


Рисунок 41 – Интерфейс общения с ботом, когда перед камерой находится неидентифицированный человек

## ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы были выполнены следующие задачи:

- рассмотрен способ идентификации пользователей на основе распознавания лиц;
- рассмотрены различные виды нейронных сетей, используемых для создания приложений-собеседников;
- реализован пользовательский интерфейс приложения;
- реализована база данных для хранения информации, необходимой для работы приложения;
- реализовано распознавание лиц и возможность ввода имен;
- реализована нейронная сеть для работы чат-бота – модель sequence to sequence с энкодером RNN GRU, декодером RNN GRU и механизмом внимания;
- реализована возможность общения с ботом в пользовательском интерфейсе;
- проведено тестирование приложения.

Задачи выполнены, цель выпускной квалификационной работы достигнута.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ши, В. Рекуррентные нейронные сети и их использование в задачах обработки естественного языка / В. Ши // Научный аспект. – 2024. – Т. 36, № 1. – С. 4714-4718.
2. Сайт //  
[https://neerc.ifmo.ru/wiki/index.php?title=%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%80%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B5\\_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5\\_%D1%81%D0%B5%D1%82%D0%B8](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%80%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B5_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D1%81%D0%B5%D1%82%D0%B8) (дата обращения 28.04.24)
3. Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.
4. Сайт //  
[https://neerc.ifmo.ru/wiki/index.php?title=%D0%94%D0%BE%D0%BB%D0%B3%D0%B0%D1%8F\\_%D0%BA%D1%80%D0%B0%D1%82%D0%BA%D0%BE%D1%81%D1%80%D0%BE%D1%87%D0%BD%D0%B0%D1%8F\\_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C](https://neerc.ifmo.ru/wiki/index.php?title=%D0%94%D0%BE%D0%BB%D0%B3%D0%B0%D1%8F_%D0%BA%D1%80%D0%B0%D1%82%D0%BA%D0%BE%D1%81%D1%80%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C) (дата обращения 28.04.24)
5. Будыльскийкий, Д. В. GRU и LSTM: современные рекуррентные нейронные сети / Д. В. Будыльскийкий // Молодой ученый. – 2015. – № 15(95). – С. 51-54.
6. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017. <https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf>
7. Митина, О. А. генерация текста с помощью нейронных сетей / О. А. Митина, В. В. Жаров // Национальная Ассоциация Ученых. – 2023. – № 90-2. – С. 19-27.
8. Sutskever I., Vinyals O., Le Q. V. Sequence to sequence learning with neural networks //Advances in neural information processing systems. – 2014. – С. 3104-3112.

9. Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).
10. Сайт // [https://pytorch.org/tutorials/beginner/chatbot\\_tutorial.html](https://pytorch.org/tutorials/beginner/chatbot_tutorial.html) (дата обращения 28.04.24)
11. Когай, И. Е. Особенности нейронной сети BERT и способы ее использования / И. Е. Когай, П. И. Маслакова, Е. В. Попова // Цифровизация экономики: направления, методы, инструменты : СБОРНИК МАТЕРИАЛОВ V ВСЕРОССИЙСКОЙ НАУЧНО-ПРАКТИЧЕСКОЙ КОНФЕРЕНЦИИ, Краснодар, 16–21 января 2023 года. – Краснодар: Кубанский государственный аграрный университет имени И.Т. Трубилина, 2023. – С. 365-367.
12. Тюрюмина, В. А. Разработка нейросети GPT-3 на базе NLP / В. А. Тюрюмина // СОВРЕМЕННЫЕ ДОСТИЖЕНИЯ МОЛОДЕЖНОЙ науки : сборник статей Международного научно-исследовательского конкурса, Петрозаводск, 11 мая 2021 года. – Петрозаводск: Международный центр научного партнерства «Новая Наука» (ИП Ивановская Ирина Игоревна), 2021. – С. 14-18.
13. Сайт PyTorch // <https://pytorch.org/> (дата обращения 16.05.24)
14. Документация библиотеки face\_recognition: сайт // <https://face-recognition.readthedocs.io/en/latest/index.html#> (дата обращения 16.05.24).
15. Описание библиотеки speech\_recognition: сайт // <https://pypi.org/project/SpeechRecognition/> (дата обращения 16.05.24).
16. Документация PostgreSQL: сайт // <https://www.postgresql.org/docs/> (дата обращения 16.05.24).

## ПРИЛОЖЕНИЕ А

В данном приложении представлен код программы.

В листинге 1 представлен код программы.

### Листинг 1 – Код программы

```
import torch
import torch.nn as nn
from torch import optim
import torch.nn.functional as F
import random
import re
import os
import itertools
import time
import cv2
import face_recognition
import speech_recognition
import psycpg2
from tkinter import *
import tkinter as tk
from tkinter import scrolledtext
from PIL import Image, ImageTk
import shutil
import pickle
import sys
```

```
host = "127.0.0.1"
user = "postgres"
password = "qwerty"
db_name = "face_recognition_chatbot"
```

```
#-----SQL queries-----
```



```

-----

def sql_query(query):
    try:
        connection = psycopg2.connect(
            host=host,
            user=user,
            password=password,
            database=db_name
        )
        connection.autocommit = True
        with connection.cursor() as cursor:
            cursor.execute(query)
    except Exception as _ex:
        print(_ex)
    finally:
        if connection:
            connection.close()

def sql_query_with_return_fetchone(query):
    try:
        connection = psycopg2.connect(
            host=host,
            user=user,
            password=password,
            database=db_name
        )
        connection.autocommit = True
        with connection.cursor() as cursor:
            cursor.execute(query)
            return cursor.fetchone()[0]
    except Exception as _ex:
        print(_ex)

```

```

finally:
    if connection:
        connection.close()

def sql_query_with_return_fetchall(query):
    try:
        connection = psycopg2.connect(
            host=host,
            user=user,
            password=password,
            database=db_name
        )
        connection.autocommit = True
        with connection.cursor() as cursor:
            cursor.execute(query)
            return cursor.fetchall()
    except Exception as _ex:
        print(_ex)
    finally:
        if connection:
            connection.close()

def create_table_images():
    query = """CREATE TABLE if not exists IMAGES (
                ImageId serial PRIMARY KEY,
                FaceEncoding BYTEA NOT NULL,
                IdentifiedPersonId integer NULL,
                CONSTRAINT fk_image_identified_person FOREIGN
KEY (IdentifiedPersonId) REFERENCES IDENTIFIED_PERSONS
(IdentifiedPersonId) ON DELETE CASCADE,
                UnidentifiedPersonId integer NULL,
                CONSTRAINT fk_image_unidentified_person
FOREIGN KEY (UnidentifiedPersonId) REFERENCES UNIDENTIFIED_PERSONS

```

```

(UnidentifiedPersonId) ON DELETE CASCADE,
        DateAndTime TIMESTAMP NOT NULL,
        CONSTRAINT check_ids CHECK
((IdentifiedPersonId IS NOT NULL AND UnidentifiedPersonId IS NULL)
or (IdentifiedPersonId IS NULL AND UnidentifiedPersonId IS NOT
NULL))
    );"""
    sql_query(query)

```

```

def create_table_identified_persons():
    query = """CREATE TABLE if not exists IDENTIFIED_PERSONS(
        IdentifiedPersonId serial PRIMARY KEY,
        Name varchar(50) UNIQUE NOT NULL,
        DateAndTimeOfFirstRecognition TIMESTAMP NOT
NULL,
        DateAndTimeOfLastRecognition TIMESTAMP NOT
NULL
    );"""
    sql_query(query)

```

```

def create_table_unidentified_persons():
    query = """CREATE TABLE if not exists UNIDENTIFIED_PERSONS(
        UnidentifiedPersonId serial PRIMARY KEY,
        DateAndTimeOfFirstRecognition TIMESTAMP NOT
NULL,
        DateAndTimeOfLastRecognition TIMESTAMP NOT
NULL
    );"""
    sql_query(query)

```

```

def create_table_dialogues():
    query = """CREATE TABLE if not exists DIALOGUES(

```

```

        DialogId serial PRIMARY KEY,
        Question varchar(200) NOT NULL,
        Answer varchar(200) NOT NULL,
        IdentifiedPersonId integer NOT NULL,
        CONSTRAINT fk_dialog_identified_person FOREIGN
KEY (IdentifiedPersonId) REFERENCES IDENTIFIED_PERSONS
(IdentifiedPersonId) ON DELETE CASCADE
    );"""

```

```

sql_query(query)

```

```

def drop_table_images():
    query = "DROP TABLE IMAGES;"
    sql_query(query)

```

```

def drop_table_identified_persons():
    query = "DROP TABLE IDENTIFIED_PERSONS;"
    sql_query(query)

```

```

def drop_table_unidentified_persons():
    query = "DROP TABLE UNIDENTIFIED_PERSONS;"
    sql_query(query)

```

```

def drop_table_dialogues():
    query = "DROP TABLE DIALOGUES;"
    sql_query(query)

```

```

def insert_identified_person_image_into_images(face_encoding,
identified_person_id):

```

```

    try:
        connection = psycopg2.connect(

```

```

        host=host,
        user=user,
        password=password,
        database=db_name
    )
    connection.autocommit = True
    with connection.cursor() as cursor:
        cursor.execute(
            """INSERT INTO IMAGES(FaceEncoding,
IdentifiedPersonId, DateAndTime) VALUES (%s, %s, NOW());""",
            (face_encoding, identified_person_id)
        )
    except Exception as _ex:
        print(_ex)
    finally:
        if connection:
            connection.close()

def insert_unidentified_person_image_into_images(face_encoding,
unidentified_person_id):
    try:
        connection = psycopg2.connect(
            host=host,
            user=user,
            password=password,
            database=db_name
        )
        connection.autocommit = True
        with connection.cursor() as cursor:
            cursor.execute(
                """INSERT INTO IMAGES(FaceEncoding,
UnidentifiedPersonId, DateAndTime) VALUES (%s, %s, NOW());""",
                (face_encoding, unidentified_person_id)
            )
    except Exception as _ex:

```

```

        print(_ex)
    finally:
        if connection:
            connection.close()

def insert_into_identified_persons(name):
    query = "INSERT INTO IDENTIFIED_PERSONS (Name,
DateAndTimeOfFirstRecognition, DateAndTimeOfLastRecognition)
VALUES ('" + name + "', NOW(), NOW());"
    sql_query(query)

def insert_into_dialogues(identified_person_id, question, answer):
    query = "INSERT INTO DIALOGUES (IdentifiedPersonId, Question,
Answer) VALUES (" + str(identified_person_id) + ", '" + question +
"', '" + answer + "');"
    sql_query(query)

def
insert_into_identified_persons_with_date_and_time_of_first_and_las
t_recognitions(name,

date_and_time_of_first_recognition,

date_and_time_of_last_recognition):
    query = "INSERT INTO IDENTIFIED_PERSONS (Name,
DateAndTimeOfFirstRecognition, DateAndTimeOfLastRecognition)
VALUES ('" + name + "', '" +
str(date_and_time_of_first_recognition) + "', '" +
str(date_and_time_of_last_recognition) + "');"
    sql_query(query)

```

```

def insert_into_unidentified_persons():
    query = "INSERT INTO
UNIDENTIFIED_PERSONS(DateAndTimeOfFirstRecognition,
DateAndTimeOfLastRecognition) VALUES (NOW(), NOW());"
    sql_query(query)

def count_images():
    query = "SELECT COUNT(*) FROM IMAGES;"
    res = sql_query_with_return_fetchone(query)
    return res

def count_identified_persons():
    query = "SELECT COUNT(*) FROM IDENTIFIED_PERSONS;"
    res = sql_query_with_return_fetchone(query)
    return res

def count_types_of_changes():
    query = "SELECT COUNT(*) FROM TYPES_OF_CHANGES;"
    res = sql_query_with_return_fetchone(query)
    return res

def count_unidentified_persons():
    query = "SELECT COUNT(*) FROM UNIDENTIFIED_PERSONS;"
    res = sql_query_with_return_fetchone(query)
    return res

def count_images_of_identified_person(identified_person_id):
    query = "SELECT COUNT(*) FROM IMAGES WHERE IdentifiedPersonId
= " + str(identified_person_id) + ";"
    res = sql_query_with_return_fetchone(query)

```

```

    return res

def count_images_of_unidentified_person(unidentified_person_id):
    query = "SELECT COUNT(*) FROM IMAGES WHERE
UnidentifiedPersonId = " + str(unidentified_person_id) + ";"
    res = sql_query_with_return_fetchone(query)
    return res

def get_id_by_name_in_table_identified_persons(name):
    query = "SELECT IdentifiedPersonId FROM IDENTIFIED_PERSONS
WHERE Name = '" + name + "'"
    res = sql_query_with_return_fetchone(query)
    return res

def
get_name_by_id_in_table_identified_persons(identified_person_id):
    query = "SELECT Name FROM IDENTIFIED_PERSONS WHERE
IdentifiedPersonId = " + str(identified_person_id) + ";"
    res = sql_query_with_return_fetchone(query)
    return res

def
find_identified_person_face_encoding_in_table_images(identified_person_id):
    query = "SELECT FaceEncoding FROM IMAGES WHERE
IdentifiedPersonId = " + str(identified_person_id) + ";"
    res = pickle.loads(sql_query_with_return_fetchone(query))
    return res

def

```



```

find_unidentified_person_face_encoding_in_table_images(unidentified_person_id):
    query = "SELECT FaceEncoding FROM IMAGES WHERE
UnidentifiedPersonId = " + str(unidentified_person_id) + ";"
    res = pickle.loads(sql_query_with_return_fetchone(query))
    return res

```

```

def is_there_this_name_in_table_identified_persons(name):
    query = "SELECT EXISTS (SELECT 1 FROM IDENTIFIED_PERSONS WHERE
Name = '" + name + "');"
    res = sql_query_with_return_fetchone(query)
    return res

```

```

def
is_there_this_id_in_table_unidentified_persons(unidentified_person_id):
    query = "SELECT EXISTS (SELECT 1 FROM UNIDENTIFIED_PERSONS
WHERE UnidentifiedPersonId = " + str(unidentified_person_id) +
");"
    res = sql_query_with_return_fetchone(query)
    return res

```

```

def
delete_from_table_unidentified_persons_by_id(unidentified_person_id):
    query = "DELETE FROM UNIDENTIFIED_PERSONS WHERE
UnidentifiedPersonId = " + str(unidentified_person_id) + ";"
    sql_query(query)

```

```

def delete_from_table_dialogues_by_id(identified_person_id):
    query = "DELETE FROM DIALOGUES WHERE IdentifiedPersonId = " +

```

```
str(identified_person_id) + ";"
    sql_query(query)
```

```
def
```

```
get_date_and_time_of_first_recognition_in_table_unidentified_persons(
    unidentified_person_id):
```

```
    query = "SELECT DateAndTimeOfFirstRecognition FROM
UNIDENTIFIED_PERSONS WHERE UnidentifiedPersonId = " +
str(unidentified_person_id) + ";"
    res = sql_query_with_return_fetchone(query)
    return res
```

```
def
```

```
get_date_and_time_of_last_recognition_in_table_unidentified_persons(
    unidentified_person_id):
```

```
    query = "SELECT DateAndTimeOfLastRecognition FROM
UNIDENTIFIED_PERSONS WHERE UnidentifiedPersonId = " +
str(unidentified_person_id) + ";"
    res = sql_query_with_return_fetchone(query)
    return res
```

```
def
```

```
get_date_and_time_of_first_recognition_in_table_identified_persons(
    identified_person_id):
```

```
    query = "SELECT DateAndTimeOfFirstRecognition FROM
IDENTIFIED_PERSONS WHERE IdentifiedPersonId = " +
str(identified_person_id) + ";"
    res = sql_query_with_return_fetchone(query)
    return res
```

```
def
```

```

get_date_and_time_of_last_recognition_in_table_identified_persons(
    identified_person_id):
    query = "SELECT DateAndTimeOfLastRecognition FROM
IDENTIFIED_PERSONS WHERE IdentifiedPersonId = " +
str(identified_person_id) + ";"
    res = sql_query_with_return_fetchone(query)
    return res

```

```

def
update_unidentified_person_id_to_identified_person_id_in_table_images(
    unidentified_person_id, identified_person_id):
    query = "UPDATE IMAGES SET IdentifiedPersonId = " +
str(identified_person_id) + ", UnidentifiedPersonId = NULL WHERE
UnidentifiedPersonId = " + str(unidentified_person_id) + ";"
    sql_query(query)

```

```

def update_name_in_identified_persons(old_name, new_name):
    query = "UPDATE IDENTIFIED_PERSONS SET Name = '" + new_name +
"' WHERE Name = '" + old_name + "';"
    sql_query(query)

```

```

def
update_date_and_time_of_last_recognition_in_table_identified_persons(
    identified_person_id):
    query = "UPDATE IDENTIFIED_PERSONS SET
DateAndTimeOfLastRecognition = NOW() WHERE IdentifiedPersonId = "
+ str(identified_person_id) + ";"
    sql_query(query)

```

```

def
update_date_and_time_of_last_recognition_in_table_unidentified_per

```

```

sons(identified_person_id):
    query = "UPDATE IDENTIFIED_PERSONS SET
DateAndTimeOfLastRecognition = NOW() WHERE IdentifiedPersonId =
" + str(identified_person_id) + ";"
    sql_query(query)

def get_ids_from_table_identified_persons():
    query = "SELECT IdentifiedPersonId FROM IDENTIFIED_PERSONS;"
    res = sql_query_with_return_fetchall(query)
    return res

def get_ids_from_table_unidentified_persons():
    query = "SELECT UnidentifiedPersonId FROM
UNIDENTIFIED_PERSONS;"
    res = sql_query_with_return_fetchall(query)
    return res

def
get_questions_and_answers_by_identified_person_id_in_table_dialogu
es(identified_person_id):
    query = "SELECT dialogues.question, dialogues.answer FROM
DIALOGUES WHERE IdentifiedPersonId = " + str(identified_person_id)
+ ";"
    res = sql_query_with_return_fetchall(query)
    return res

#-----Speech recognition-----
-----

def listen():

```

```

try:
    with speech_recognition.Microphone() as mic:
        sr.adjust_for_ambient_noise(source=mic, duration=0.5)
        audio = sr.listen(source=mic)
        query = sr.recognize_google(audio_data=audio,
language='ru-RU').lower()
        return query
except speech_recognition.UnknownValueError:
    return None

#-----Delete dataset-----
-----

def delete_dataset():
    folders = os.listdir("dataset")
    for folder in folders:
        shutil.rmtree("dataset/" + folder)

    drop_table_images()
    drop_table_dialogues()
    drop_table_identified_persons()
    drop_table_unidentified_persons()

    create_table_identified_persons()
    create_table_unidentified_persons()
    create_table_images()
    create_table_dialogues()

#-----Changing names-----
-----

```

```

def is_name_belongs_to_unidentified_person(name):
    return name[:6] == "person" and name[6:].isdigit() and
is_there_this_id_in_table_unidentified_persons(int(name[6:]))

def is_name_belongs_to_identified_person(name):
    return is_there_this_name_in_table_identified_persons(name)

def change_name(old_name, new_name):
    if is_name_belongs_to_identified_person(old_name) and not
(is_name_belongs_to_identified_person(new_name)) and not (
    is_name_belongs_to_unidentified_person(new_name)) and new_name
!= "":
        update_name_in_identified_persons(old_name, new_name)
        dialog_label1.config(text="Имя " + old_name + " изменено
на " + new_name, width="50")
        dialog_label1.update_idletasks()
        if os.path.isdir("dataset/" + old_name):
            os.rename("dataset/" + old_name, "dataset/" +
new_name)
        elif is_name_belongs_to_unidentified_person(old_name) and not
(
    is_name_belongs_to_identified_person(new_name)) and not (
    is_name_belongs_to_unidentified_person(new_name)) and new_name
!= "":
            unidentified_person_id = int(old_name[6:])
            date_and_time_of_first_recognition =
get_date_and_time_of_first_recognition_in_table_unidentified_perso
ns(
    unidentified_person_id)
            date_and_time_of_last_recognition =
get_date_and_time_of_last_recognition_in_table_unidentified_person
s(
    unidentified_person_id)

```

```

insert_into_identified_persons_with_date_and_time_of_first_and_last_recognitions(new_name,

date_and_time_of_first_recognition,

date_and_time_of_last_recognition)
        identified_person_id = count_identified_persons()

update_unidentified_person_id_to_identified_person_id_in_table_images(unidentified_person_id,

identified_person_id)

delete_from_table_unidentified_persons_by_id(unidentified_person_id)

        dialog_label1.config(text="Имя " + old_name + " изменено
на " + new_name, width="50")
        dialog_label1.update_idletasks()
        if os.path.isdir("dataset/" + old_name):
            os.rename("dataset/" + old_name, "dataset/" +
new_name)
        else:
            dialog_label1.config(text="Имена введены некорректно",
width="50")
            dialog_label1.update_idletasks()

def change_name_from_speech_recognition():
    global name_now, number_of_persons_in_front_of_the_camera
    if number_of_persons_in_front_of_the_camera == 0:
        dialog_label1.config(text='Перед камерой никого нет',
width="50")
        dialog_label1.update_idletasks()
        return

```

```

        if number_of_persons_in_front_of_the_camera > 1:
            dialog_label1.config(text='Перед камерой находится более
одного человека', width="50")
            dialog_label1.update_idletasks()
            return
        dialog_label1.config(text='Скажите имя', width="50")
        dialog_label1.update_idletasks()
        query = listen()
        if query:
            change_name(name_now, query[0].upper() + query[1:])
        else:
            dialog_label1.config(text='Не получилось распознать имя',
width="50")
            dialog_label1.update_idletasks()

def change_name_from_input_person_name_in_front_of_camera():
    global input_person_name_in_front_of_camera, dialog_label1,
number_of_persons_in_front_of_the_camera
    if number_of_persons_in_front_of_the_camera == 0:
        dialog_label1.config(text='Перед камерой никого нет',
width="50")
        dialog_label1.update_idletasks()
        return
    if number_of_persons_in_front_of_the_camera > 1:
        dialog_label1.config(text='Перед камерой находится более
одного человека', width="50")
        dialog_label1.update_idletasks()
        return
    old_name = name_now
    new_name = input_person_name_in_front_of_camera.get()
    change_name(old_name, new_name)

def change_name_from_input_now_name_and_input_new_name():

```



```

global input_now_name, input_new_name
old_name = input_now_name.get()
new_name = input_new_name.get()
change_name(old_name, new_name)

def get_identified_person_names():
    identified_person_ids =
get_ids_from_table_identified_persons()
    identified_person_names = []
    for i in range(len(identified_person_ids)):
        identified_person_id = identified_person_ids[i][0]

    identified_person_names.append(get_name_by_id_in_table_identified_
persons(identified_person_id))
    return identified_person_names

#-----Output dialog-----
-----

def fill_dialog(name):
    global interface2_scroll
    delete_dialog_in_scroll_text()
    identified_person_id =
get_id_by_name_in_table_identified_persons(name)
    identified_person_id_dialogues =
get_questions_and_answers_by_identified_person_id_in_table_dialogu
es(identified_person_id)
    for i in range(len(identified_person_id_dialogues)):
        question = identified_person_id_dialogues[i][0]
        answer = identified_person_id_dialogues[i][1]
        interface2_scroll.insert("end", name + ": " + question +
"\n")

```

```

        interface2_scroll.insert("end", "Bot: " + answer + "\n")
    interface2_scroll.see("end")

def delete_dialog_in_scroll_text():
    global interface2_scroll
    interface2_scroll.delete('1.0', 'end')

def delete_dialog():
    global names_persons_in_front_of_the_camera_now
    if len(names_persons_in_front_of_the_camera_now) == 1:
        identified_person_id =
get_id_by_name_in_table_identified_persons(names_persons_in_front_
of_the_camera_now[0])
        delete_from_table_dialogues_by_id(identified_person_id)
        delete_dialog_in_scroll_text()

#-----Face recognition-----
-----

def face_rec():
    global last_time, new_time, name_now,
number_of_persons_in_front_of_the_camera,
names_persons_in_front_of_the_camera_now,
names_persons_in_front_of_the_camera_last, interface2_label
    success, image = cap.read()
    locations = face_recognition.face_locations(image)
    encodings = face_recognition.face_encodings(image, locations)
    is_3_sec_passed = False
    number_of_persons_in_front_of_the_camera = len(locations)
    names_persons_in_front_of_the_camera_now = []
    for face_location, face_encoding in zip(locations, encodings):

```

```

        name = ""
        recognized_identified_person_id = 0
        recognized_unidentified_person_id = 0
        identified_person_ids =
get_ids_from_table_identified_persons()
        unidentified_person_ids =
get_ids_from_table_unidentified_persons()
        if identified_person_ids:
            identified_person_number = len(identified_person_ids)
        else:
            identified_person_number = 0
        if unidentified_person_ids:
            unidentified_person_number =
len(unidentified_person_ids)
        else:
            unidentified_person_number = 0
        for i in range(identified_person_number):
            identified_person_id = identified_person_ids[i][0]
            identified_person_face_encoding =
find_identified_person_face_encoding_in_table_images(identified_per
rson_id)
            if face_recognition.compare_faces([face_encoding],
identified_person_face_encoding)[0]:
                recognized_identified_person_id =
identified_person_id
                name =
get_name_by_id_in_table_identified_persons(identified_person_id)
                break
        for i in range(unidentified_person_number):
            unidentified_person_id = unidentified_person_ids[i][0]
            unidentified_person_face_encoding =
find_unidentified_person_face_encoding_in_table_images(
unidentified_person_id)
            if face_recognition.compare_faces([face_encoding],
unidentified_person_face_encoding)[0]:

```

```

        recognized_unidentified_person_id =
unidentified_person_id
        name = "person" +
str(recognized_unidentified_person_id)
        break
    top, right, bottom, left = face_location
    left_top = (left, top)
    right_bottom = (right, bottom)
    color = [255, 0, 0]
    cv2.rectangle(image, left_top, right_bottom, color, 4)
    new_time = time.time()
    if new_time - last_time >= 3:
        is_3_sec_passed = True
        face_img = image[top:bottom, left:right]
        face_img = cv2.cvtColor(face_img, cv2.COLOR_BGR2RGB)
        pil_img = Image.fromarray(face_img)
        if recognized_identified_person_id != 0:
            count =
count_images_of_identified_person(recognized_identified_person_id)
+ 1
            pil_img.save("dataset/" + name + "/"
f"img_{count}.jpg")

insert_identified_person_image_into_images(pickle.dumps(face_encod
ing), recognized_identified_person_id)

update_date_and_time_of_last_recognition_in_table_identified_perso
ns(recognized_identified_person_id)
        elif recognized_unidentified_person_id != 0:
            count =
count_images_of_unidentified_person(recognized_unidentified_person
_id) + 1
            pil_img.save("dataset/" + name + "/"
f"img_{count}.jpg")

```

```

insert_unidentified_person_image_into_images(pickle.dumps(face_encoding),

recognized_unidentified_person_id)

update_date_and_time_of_last_recognition_in_table_unidentified_persons(

    recognized_unidentified_person_id)
else:
    count = identified_person_number +
unidentified_person_number + 1
    name = f"person{count}"
    os.makedirs("dataset/" + name)
    pil_img.save("dataset/" + name + "/" +
f"img_1.jpg")
    insert_into_unidentified_persons()

insert_unidentified_person_image_into_images(pickle.dumps(face_encoding), count)
    left_bottom = (left, bottom)
    right_bottom = (right, bottom + 20)
    name_now = name
    names_persons_in_front_of_the_camera_now.append(name)
    cv2.rectangle(image, left_bottom, right_bottom, color,
cv2.FILLED)
    cv2.putText(image, name, (left + 10, bottom + 15),
cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 4)
    if is_3_sec_passed:
        last_time = new_time

    identified_person_names = get_identified_person_names()

    if len(names_persons_in_front_of_the_camera_now) == 0 and
len(names_persons_in_front_of_the_camera_last) != 0:
        interface2_label.config(text="Перед камерой никого нет")

```

```

        delete_dialog_in_scroll_text()

    elif (len(names_persons_in_front_of_the_camera_now) ==
len(names_persons_in_front_of_the_camera_last) == 1 and
names_persons_in_front_of_the_camera_now[0] !=
names_persons_in_front_of_the_camera_last[0] and
names_persons_in_front_of_the_camera_now[0] in
identified_person_names) or
(len(names_persons_in_front_of_the_camera_last) != 1 and
len(names_persons_in_front_of_the_camera_now) == 1 and
names_persons_in_front_of_the_camera_now[0] in
identified_person_names):
        interface2_label.config(text="")
        fill_dialog(names_persons_in_front_of_the_camera_now[0])
    elif (len(names_persons_in_front_of_the_camera_now) ==
len(names_persons_in_front_of_the_camera_last) == 1 and
names_persons_in_front_of_the_camera_now[0] !=
names_persons_in_front_of_the_camera_last[0] and
names_persons_in_front_of_the_camera_now[0] not in
identified_person_names) or
(len(names_persons_in_front_of_the_camera_last) != 1 and
len(names_persons_in_front_of_the_camera_now) == 1 and
names_persons_in_front_of_the_camera_now[0] not in
identified_person_names):
        interface2_label.config(text="Перед камерой находится
неидентифицированный человек")
        delete_dialog_in_scroll_text()
    elif len(names_persons_in_front_of_the_camera_now) > 1:
        interface2_label.config(text="Перед камерой находится
более одного человека")
        delete_dialog_in_scroll_text()

    names_persons_in_front_of_the_camera_last =
names_persons_in_front_of_the_camera_now[:]

    frame = image

```

```

cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
img = Image.fromarray(cv2image)
imgtk = ImageTk.PhotoImage(image=img)
main_label.imgtk = imgtk
main_label.configure(image=imgtk)
main_label.after(10, face_rec)

def close_program():
    sys.exit()

if not os.path.isdir("dataset"):
    os.makedirs("dataset")

create_table_identified_persons()
create_table_unidentified_persons()
create_table_images()
create_table_dialogues()

last_time = time.time()
new_time = time.time()

name_now = ""
number_of_persons_in_front_of_the_camera = 0

names_persons_in_front_of_the_camera_now = []
names_persons_in_front_of_the_camera_last = []

is_dialog_window_on = False

sr = speech_recognition.Recognizer()
sr.pause_threshold = 0.5

```

```
#-----Chatbot-----  
-----
```

```
USE_CUDA = torch.cuda.is_available()  
device = torch.device("cuda" if USE_CUDA else "cpu")
```

```
PAD_token = 0  
SOS_token = 1  
EOS_token = 2
```

```
class Dictionary:  
    def __init__(self):  
        self.word_to_index = {}  
        self.word_to_count = {}  
        self.index_to_word = {PAD_token: "PAD", SOS_token: "SOS",  
EOS_token: "EOS"}  
        self.num_words = 3  
  
    def add_sentence(self, sentence):  
        for word in sentence.split(' '):  
            self.add_word(word)  
  
    def add_word(self, word):  
        if word not in self.word_to_index:  
            self.word_to_index[word] = self.num_words  
            self.word_to_count[word] = 1  
            self.index_to_word[self.num_words] = word  
            self.num_words += 1  
        else:  
            self.word_to_count[word] += 1
```



```
MAX_LENGTH = 10
```

```
def prepare_string(s):
    s = s.lower().strip()
    s = re.sub(r"([.!?])", r" \1", s)
    s = re.sub(r"^[a-zA-Я.!?]+", r" ", s)
    s = re.sub(r"\s+", r" ", s).strip()
    return s

# pairs = []
#
# max_number_of_pairs = 1000000
#
# with open('dialogues1.txt', 'r') as file1:
#     lines1 = file1.readlines()
#
#
# count = 0
#
# for i in range(len(lines1) - 1):
#     lines1[i] = lines1[i].strip()
#     lines1[i + 1] = lines1[i + 1].strip()
#     if len(lines1[i]) > 0 and len(lines1[i + 1]) > 0 and
# lines1[i][0] == "-" and lines1[i + 1][0] == "-":
#         pairs.append([prepare_string(lines1[i][2:]),
# prepare_string(lines1[i + 1][2:])])
#         count += 1
#         if count >= max_number_of_pairs:
#             break
#
# with open('dialogues2.txt', 'r') as file2:
#     lines2 = file2.readlines()
```

```

#
# for i in range(len(lines2) - 1):
#     lines2[i] = lines2[i].strip()
#     lines2[i + 1] = lines2[i + 1].strip()
#     if len(lines2[i]) > 0 and len(lines2[i + 1]) > 0 and
lines2[i][0] == "-" and lines2[i + 1][0] == "-":
#         pairs.append([prepare_string(lines2[i][2:]),
prepare_string(lines2[i + 1][2:])])
#         count += 1
#         if count >= max_number_of_pairs:
#             break

```

```

def pairs_smaller_than_max_len(pairs):
    used_pairs = []
    for pair in pairs:
        if len(pair[0].split(' ')) < MAX_LENGTH and
len(pair[1].split(' ')) < MAX_LENGTH:
            used_pairs.append(pair)
    return used_pairs

```

```

# pairs = pairs_smaller_than_max_len(pairs)

```

```

words_dict = Dictionary()

```

```

# for pair in pairs:
#     words_dict.add_sentence(pair[0])
#     words_dict.add_sentence(pair[1])

```

```

MIN_COUNT = 3

```

```

def trim_rare_words(words_dict, pairs, MIN_COUNT):
    used_words = []

```

```

for k, v in words_dict.word_to_count.items():
    if v >= MIN_COUNT:
        used_words.append(k)

words_dict.word_to_index = {}
words_dict.word_to_count = {}
words_dict.index_to_word = {PAD_token: "PAD", SOS_token:
"SOS", EOS_token: "EOS"}
words_dict.num_words = 3

for word in used_words:
    words_dict.add_word(word)

used_pairs = []
for pair in pairs:
    input_sentence = pair[0]
    output_sentence = pair[1]
    use_input_sentence = True
    use_output_sentence = True
    for word in input_sentence.split(' '):
        if word not in used_words:
            use_input_sentence = False
            break
    for word in output_sentence.split(' '):
        if word not in used_words:
            use_output_sentence = False
            break
    if use_input_sentence and use_output_sentence:
        used_pairs.append(pair)

return used_pairs

# pairs = trim_rare_words(words_dict, pairs, MIN_COUNT)

```

```

def indexes_from_sentence(words_dict, sentence):
    return [words_dict.word_to_index[word] for word in
sentence.split(' ')] + [EOS_token]

def zero_padding(l):
    return list(itertools.zip_longest(*l, fillvalue=PAD_token))

def get_binary_matrix(l):
    m = []
    for i, seq in enumerate(l):
        m.append([])
        for token in seq:
            if token == PAD_token:
                m[i].append(0)
            else:
                m[i].append(1)
    return m

def input_var(l, words_dict):
    indexes_batch = [indexes_from_sentence(words_dict, sentence)
for sentence in l]
    lengths = torch.tensor([len(indexes) for indexes in
indexes_batch])
    pad_list = zero_padding(indexes_batch)
    pad_var = torch.LongTensor(pad_list)
    return pad_var, lengths

def output_var(l, words_dict):
    indexes_batch = [indexes_from_sentence(words_dict, sentence)

```

```

for sentence in l]
    max_target_len = max([len(indexes) for indexes in
indexes_batch])
    pad_list = zero_padding(indexes_batch)
    mask = get_binary_matrix(pad_list)
    mask = torch.BoolTensor(mask)
    pad_var = torch.LongTensor(pad_list)
    return pad_var, mask, max_target_len

def batch_to_train_data(words_dict, pair_batch):
    pair_batch.sort(key=lambda x: len(x[0].split(" ")),
reverse=True)
    input_batch, output_batch = [], []
    for pair in pair_batch:
        input_batch.append(pair[0])
        output_batch.append(pair[1])
    inp, lengths = input_var(input_batch, words_dict)
    output, mask, max_target_len = output_var(output_batch,
words_dict)
    return inp, lengths, output, mask, max_target_len

class EncoderRNN(nn.Module):
    def __init__(self, hidden_size, embedding, n_layers=1,
dropout=0):
        super(EncoderRNN, self).__init__()
        self.n_layers = n_layers
        self.hidden_size = hidden_size
        self.embedding = embedding
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers,
dropout=(0 if n_layers == 1 else dropout), bidirectional=True)

    def forward(self, input_seq, input_lengths, hidden=None):
        embedded = self.embedding(input_seq)

```

```

        packed = nn.utils.rnn.pack_padded_sequence(embedded,
input_lengths)
        outputs, hidden = self.gru(packed, hidden)
        outputs, _ = nn.utils.rnn.pad_packed_sequence(outputs)
        outputs = outputs[:, :, :self.hidden_size] + outputs[:, :
, self.hidden_size:]
        return outputs, hidden

```

```

class Attn(nn.Module):
    def __init__(self, hidden_size):
        super(Attn, self).__init__()
        self.hidden_size = hidden_size

    def dot_score(self, hidden, encoder_output):
        return torch.sum(hidden * encoder_output, dim=2)

    def forward(self, hidden, encoder_outputs):
        attn_energies = self.dot_score(hidden, encoder_outputs)
        attn_energies = attn_energies.t()
        return F.softmax(attn_energies, dim=1).unsqueeze(1)

```

```

class DecoderRNN(nn.Module):
    def __init__(self, embedding, hidden_size, output_size,
n_layers=1, dropout=0.1):
        super(DecoderRNN, self).__init__()

        self.hidden_size = hidden_size
        self.output_size = output_size
        self.n_layers = n_layers
        self.dropout = dropout

        self.embedding = embedding
        self.embedding_dropout = nn.Dropout(dropout)

```

```

        self.gru = nn.GRU(hidden_size, hidden_size, n_layers,
dropout=(0 if n_layers == 1 else dropout))
        self.concat = nn.Linear(hidden_size * 2, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

        self.attn = Attn(hidden_size)

    def forward(self, input_step, last_hidden, encoder_outputs):
        embedded = self.embedding(input_step)
        embedded = self.embedding_dropout(embedded)
        rnn_output, hidden = self.gru(embedded, last_hidden)
        attn_weights = self.attn(rnn_output, encoder_outputs)
        context = attn_weights.bmm(encoder_outputs.transpose(0,
1))

        rnn_output = rnn_output.squeeze(0)
        context = context.squeeze(1)
        concat_input = torch.cat((rnn_output, context), 1)
        concat_output = torch.tanh(self.concat(concat_input))
        output = self.out(concat_output)
        output = F.softmax(output, dim=1)
        return output, hidden

def maskNLLLoss(inp, target, mask):
    nTotal = mask.sum()
    crossEntropy = -torch.log(torch.gather(inp, 1, target.view(-1,
1)).squeeze(1))
    loss = crossEntropy.masked_select(mask).mean()
    loss = loss.to(device)
    return loss, nTotal.item()

def train(input_variable, lengths, target_variable, mask,
max_target_len, encoder, decoder, embedding, encoder_optimizer,
decoder_optimizer, batch_size, clip, max_length=MAX_LENGTH):

```

```

encoder_optimizer.zero_grad()
decoder_optimizer.zero_grad()

input_variable = input_variable.to(device)
target_variable = target_variable.to(device)
mask = mask.to(device)
lengths = lengths.to("cpu")

loss = 0
print_losses = []
n_totals = 0

encoder_outputs, encoder_hidden = encoder(input_variable,
lengths)

decoder_input = torch.LongTensor([[SOS_token for _ in
range(batch_size)]])
decoder_input = decoder_input.to(device)

decoder_hidden = encoder_hidden[:decoder.n_layers]

use_teacher_forcing = True if random.random() <
teacher_forcing_ratio else False

if use_teacher_forcing:
    for t in range(max_target_len):
        decoder_output, decoder_hidden =
decoder(decoder_input, decoder_hidden, encoder_outputs)
        decoder_input = target_variable[t].view(1, -1)
        mask_loss, nTotal = maskNLLLoss(decoder_output,
target_variable[t], mask[t])
        loss += mask_loss
        print_losses.append(mask_loss.item() * nTotal)
        n_totals += nTotal

```



```

else:
    for t in range(max_target_len):
        decoder_output, decoder_hidden =
decoder(decoder_input, decoder_hidden, encoder_outputs)
        _, topi = decoder_output.topk(1)
        decoder_input = torch.LongTensor([[topi[i][0] for i in
range(batch_size)]])
        decoder_input = decoder_input.to(device)
        mask_loss, nTotal = maskNLLLoss(decoder_output,
target_variable[t], mask[t])
        loss += mask_loss
        print_losses.append(mask_loss.item() * nTotal)
        n_totals += nTotal

loss.backward()

_ = nn.utils.clip_grad_norm_(encoder.parameters(), clip)
_ = nn.utils.clip_grad_norm_(decoder.parameters(), clip)

encoder_optimizer.step()
decoder_optimizer.step()

return sum(print_losses) / n_totals

def train_iters(model_name, words_dict, pairs, encoder, decoder,
encoder_optimizer, decoder_optimizer, embedding, encoder_n_layers,
decoder_n_layers, save_dir, n_iteration, batch_size, print_every,
save_every, clip, corpus_name, load_filename):

    training_batches = [batch_to_train_data(words_dict,
[random.choice(pairs) for _ in range(batch_size)]) for _ in
range(n_iteration)]

    start_iteration = 1

```

```

print_loss = 0
if load_filename:
    start_iteration = checkpoint['iteration'] + 1

for iteration in range(start_iteration, n_iteration + 1):
    training_batch = training_batches[iteration - 1]
    input_variable, lengths, target_variable, mask,
max_target_len = training_batch

    loss = train(input_variable, lengths, target_variable,
mask, max_target_len, encoder, decoder, embedding,
encoder_optimizer, decoder_optimizer, batch_size, clip)
    print_loss += loss

    if iteration % print_every == 0:
        print_loss_avg = print_loss / print_every
        print("Итерация: {}; Среднее значение функции потерь:
 {:.4f}".format(iteration, print_loss_avg))
        print_loss = 0

    if (iteration % save_every == 0):
        directory = os.path.join(save_dir, model_name,
corpus_name, '{}-{}_{}'.format(encoder_n_layers, decoder_n_layers,
hidden_size))
        if not os.path.exists(directory):
            os.makedirs(directory)
        torch.save({
            'iteration': iteration,
            'en': encoder.state_dict(),
            'de': decoder.state_dict(),
            'en_opt': encoder_optimizer.state_dict(),
            'de_opt': decoder_optimizer.state_dict(),
            'loss': loss,
            'words_dict_dict': words_dict.__dict__,
            'embedding': embedding.state_dict()

```

```

        }, os.path.join(directory,
'{}_{}.tar'.format(iteration, 'checkpoint'))))

class Decoder(nn.Module):
    def __init__(self, encoder, decoder):
        super(Decoder, self).__init__()
        self.encoder = encoder
        self.decoder = decoder

    def forward(self, input_seq, input_length, max_length):
        encoder_outputs, encoder_hidden = self.encoder(input_seq,
input_length)
        decoder_hidden = encoder_hidden[:decoder.n_layers]
        decoder_input = torch.ones(1, 1, device=device,
dtype=torch.long) * SOS_token
        all_tokens = torch.zeros([0], device=device,
dtype=torch.long)
        all_scores = torch.zeros([0], device=device)
        for _ in range(max_length):
            decoder_output, decoder_hidden =
self.decoder(decoder_input, decoder_hidden, encoder_outputs)
            decoder_scores, decoder_input =
torch.max(decoder_output, dim=1)
            all_tokens = torch.cat((all_tokens, decoder_input),
dim=0)
            all_scores = torch.cat((all_scores, decoder_scores),
dim=0)
            decoder_input = torch.unsqueeze(decoder_input, 0)
        return all_tokens, all_scores

def evaluate(encoder, decoder, searcher, words_dict, sentence,
max_length=MAX_LENGTH):
    indexes_batch = [indexes_from_sentence(words_dict, sentence)]

```

```

        lengths = torch.tensor([len(indexes) for indexes in
indexes_batch])
        input_batch = torch.LongTensor(indexes_batch).transpose(0, 1)
        input_batch = input_batch.to(device)
        lengths = lengths.to("cpu")
        tokens, scores = searcher(input_batch, lengths, max_length)
        decoded_words = [words_dict.index_to_word[token.item()] for
token in tokens]
        return decoded_words

```

```

model_name = 'model'

```

```

hidden_size = 500

```

```

encoder_n_layers = 2

```

```

decoder_n_layers = 2

```

```

dropout = 0.1

```

```

batch_size = 64

```

```

checkpoint_iter = 100000

```

```

# load_filename = None

```

```

load_filename = os.path.join(model_name,
                              '{}-{}_{}'.format(encoder_n_layers,
decoder_n_layers, hidden_size),

```

```

'{}_checkpoint.tar'.format(checkpoint_iter))

```

```

if load_filename:

```

```

    checkpoint = torch.load(load_filename,
map_location=torch.device('cpu'))

```

```

    encoder_sd = checkpoint['en']

```

```

    decoder_sd = checkpoint['de']

```

```

    encoder_optimizer_sd = checkpoint['en_opt']

```

```

    decoder_optimizer_sd = checkpoint['de_opt']

```

```

    embedding_sd = checkpoint['embedding']

```

```

words_dict.__dict__ = checkpoint['words_dict_dict']

embedding = nn.Embedding(words_dict.num_words, hidden_size)
if load_filename:
    embedding.load_state_dict(embedding_sd)

encoder = EncoderRNN(hidden_size, embedding, encoder_n_layers,
                    dropout)
decoder = DecoderRNN(embedding, hidden_size, words_dict.num_words,
                    decoder_n_layers, dropout)
if load_filename:
    encoder.load_state_dict(encoder_sd)
    decoder.load_state_dict(decoder_sd)

encoder = encoder.to(device)
decoder = decoder.to(device)

clip = 50.0
teacher_forcing_ratio = 1.0
learning_rate = 0.0001
decoder_learning_ratio = 5.0
n_iteration = 100000
print_every = 1
save_every = 10000

encoder.train()
decoder.train()

encoder_optimizer = optim.Adam(encoder.parameters(),
                                lr=learning_rate)
decoder_optimizer = optim.Adam(decoder.parameters(),
                                lr=learning_rate * decoder_learning_ratio)
if load_filename:

```

```

encoder_optimizer.load_state_dict(encoder_optimizer_sd)
decoder_optimizer.load_state_dict(decoder_optimizer_sd)

# for state in encoder_optimizer.state.values():
#     for k, v in state.items():
#         if isinstance(v, torch.Tensor):
#             state[k] = v.cuda()
#
# for state in decoder_optimizer.state.values():
#     for k, v in state.items():
#         if isinstance(v, torch.Tensor):
#             state[k] = v.cuda()

# train_iters(model_name, words_dict, pairs, encoder, decoder,
encoder_optimizer, decoder_optimizer,
#             embedding, encoder_n_layers, decoder_n_layers, '',
n_iteration, batch_size,
#             print_every, save_every, clip, '', load_filename)

encoder.eval()
decoder.eval()

searcher = Decoder(encoder, decoder)

#-----Interface-----
-----

root = tk.Tk()
root.title("Face recognition Chatbot")

main_label = tk.Label(root)

```

```
main_label.pack()
```

```
interface_btn_frame = tk.Frame(root)
interface_btn_frame.pack()
interfacel_btn = tk.Button(interface_btn_frame, text="Ввод имен")
interfacel_btn.pack(side="left")
interface2_btn = tk.Button(interface_btn_frame, text="Общение с
чат-ботом")
interface2_btn.pack(side="left")
```

```
interfacel_frame = tk.Frame(root)
dialog_label1 = tk.Label(interfacel_frame, text="")
dialog_label1.pack()
input_person_name_in_front_of_camera = tk.Entry(interfacel_frame)
input_person_name_in_front_of_camera.pack()
interfacel_name_btn = tk.Button(interfacel_frame, text="Ввести
имя",
command=change_name_from_input_person_name_in_front_of_camera)
interfacel_name_btn.pack()
interfacel_voice_btn = tk.Button(interfacel_frame, text="Голосовой
ввод имени", command=change_name_from_speech_recognition)
interfacel_voice_btn.pack()
interfacel_previous_label = tk.Label(interfacel_frame,
text="Текущее имя:")
interfacel_previous_label.pack()
input_now_name = tk.Entry(interfacel_frame)
input_now_name.pack()
interfacel_new_label = tk.Label(interfacel_frame, text="Новое
имя:")
interfacel_new_label.pack()
input_new_name = tk.Entry(interfacel_frame)
input_new_name.pack()
interfacel_submit_btn = tk.Button(interfacel_frame, text="Ввести
```

```

имя", command=change_name_from_input_now_name_and_input_new_name)
interface1_submit_btn.pack()
interface1_clear_btn = tk.Button(interface1_frame, text="Очистить
базу данных", command=delete_dataset)
interface1_clear_btn.pack()

interface2_frame = tk.Frame(root)
interface2_label = tk.Label(interface2_frame, text="")
interface2_label.pack()
interface2_scroll = scrolledtext.ScrolledText(interface2_frame,
height=10)
interface2_scroll.pack()
interface2_entry = tk.Entry(interface2_frame)
interface2_entry.pack()
interface2_send_btn = tk.Button(interface2_frame, text="Ввести
сообщение")
interface2_send_btn.pack()
interface2_voice_btn = tk.Button(interface2_frame, text="Ввести
сообщение голосом")
interface2_voice_btn.pack()
interface2_clear_btn = tk.Button(interface2_frame, text="Очистить
диалог", command=delete_dialog)
interface2_clear_btn.pack()

interface1_frame.pack()

cap = cv2.VideoCapture(0)
face_rec()

def show_interface1():
    global dialog_label1, input_person_name_in_front_of_camera,
input_now_name, input_new_name
    dialog_label1.config(text="")

```



```

input_person_name_in_front_of_camera.delete(0, 'end')
input_now_name.delete(0, 'end')
input_new_name.delete(0, 'end')
interface2_frame.pack_forget()
interface1_frame.pack()

def show_interface2():
    interface1_frame.pack_forget()
    interface2_frame.pack()

def send_message():

    global names_persons_in_front_of_the_camera_now

    names_identified_persons = get_identified_person_names()

    if len(names_persons_in_front_of_the_camera_now) == 1 and
names_persons_in_front_of_the_camera_now[0] in
names_identified_persons:
        message = interface2_entry.get()

        interface2_entry.delete(0, "end")

        interface2_scroll.insert("end", name_now + ": " + message
+ "\n")

        input_sentence = message
        try:
            input_sentence = prepare_string(input_sentence)
            output_words = evaluate(encoder, decoder, searcher,
words_dict, input_sentence)
            output_words[:] = [x for x in output_words if not (x
== 'EOS' or x == 'PAD')]

```

```

        identified_person_id_now =
get_id_by_name_in_table_identified_persons(name_now)
        insert_into_dialogues(identified_person_id_now,
message, ' '.join(output_words))

        interface2_scroll.insert("end", "Бот: " + '
'.join(output_words) + "\n")

    except KeyError:
        interface2_scroll.insert("end", "Бот: Я не знаю\n")

    interface2_scroll.see("end")

def send_message_voice():
    global names_persons_in_front_of_the_camera_now

    names_identified_persons = get_identified_person_names()

    if len(names_persons_in_front_of_the_camera_now) == 1 and
names_persons_in_front_of_the_camera_now[0] in
names_identified_persons:
        message = listen()

        interface2_entry.delete(0, "end")

        interface2_scroll.insert("end", name_now + ": " + message
+ "\n")

    input_sentence = message
    try:
        input_sentence = prepare_string(input_sentence)
        output_words = evaluate(encoder, decoder, searcher,
words_dict, input_sentence)

```

```

        output_words[:] = [x for x in output_words if not (x
== 'EOS' or x == 'PAD')]

        identified_person_id_now =
get_id_by_name_in_table_identified_persons(name_now)
        insert_into_dialogues(identified_person_id_now,
message, ' '.join(output_words))

        interface2_scroll.insert("end", "Бот: " + '
'.join(output_words) + "\n")

    except KeyError:
        interface2_scroll.insert("end", "Бот: Я не знаю\n")

    interface2_scroll.see("end")

interface1_btn.config(command=show_interface1)
interface2_btn.config(command=show_interface2)
interface2_send_btn.config(command=send_message)
interface2_voice_btn.config(command=send_message_voice)

root.mainloop()

```