1. НАЗНАЧЕНИЕ ПРОГРАММЫ

Специальное программное обеспечение «3i Speaker ID SDK» предназначено для решения задач распознавания речи, текстонезависимой идентификации личности диктора по голосу. 3i Speaker ID SDK используется в качестве программного обеспечения, предоставляя разработчику соответствующий функционал API.

API (сокр. англ. Application Programming Interface, интерфейс программирования приложений, интерфейс прикладного программирования) – набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением для использования во внешних программных продуктах.

К основным преимуществам приложения, предоставляющего DLL API в качестве инструмента доступа к функциональным возможностям, можно отнести:

- надёжность (за счет отсутствия необходимости взаимодействия с удаленным модулем);
- производительность (за счет использования кэша и подгрузки кода и данных в пространство адресов приложения);
- прозрачность системы взаимодействия;
- легкость внесения изменений;
- масштабируемость.

Продукт реализован в виде динамической библиотеки (DLL), написанной на языке C++. Допускается использование в нескольких параллельных потоках.

2. УСЛОВИЯ ПРИМЕНЕНИЯ

2.1. Для функционирования 3i Speaker ID SDK необходима вычислительная система с параметрами не хуже:

- CPU Intel Core i7 – 5820K 3.3 ГГц (6 физических вычислительных ядер);
- ОЗУ 16 ГБ;
- 100 Гб свободного места на жёстком диске.

2.2. Для функционирования 3i Speaker ID SDK на вычислительной системе должно быть установлено следующее общее программное обеспечение:

- операционная система – Microsoft Windows 7 SP1 или выше;
- распространяемый пакет Microsoft VC 2015 Redist (x64).

3. ОПИСАНИЕ ЗАДАЧИ

3.1 Технология работы с речевыми данными

Технология идентификации основана на применении глубоких нейронных сетей, при помощи которых по речевому сигналу генерируется вектор голосовых признаков. Для получения вектора диктора достаточно 5-ти секунд речи. Полученный вектор представляет собой модель голоса, состоит из действительных чисел, размерность вектора 112 элементов.

Для решения задач идентификации в библиотеке предусмотрена функция сравнения двух векторов при помощи косинусной меры близости. Для принятия решений рекомендуется использовать пороговое значение косинусной близости, при котором достигается минимальная суммарная ошибка.

В библиотеку встроен детектор речи, позволяющий эффективно «отсеивать» неречевые составляющие входного сигнала.

3.2 Качество идентификации

На рисунках 2.1 - 2.2 представлены графики значений ошибок первого и второго рода (FRR и FAR соответственно) в условиях работы в телефонном канале связи (phone).

Равенство ошибок достигается при пороге t=0.72. В этой точке равновероятная ошибка EER=3.39%.

FAR (False Alarm Ratio) – ошибка ложного срабатывания («чужой» диктор принят за «своего», либо диктор из числа «своих» был перепутан с иным «целевым» диктором).

FRR (False Rejection Ratio) – ошибка ложного пропуска «своего» диктора.

EER (Equal Error Rate) – равновероятная ошибка.

Mean Error – средняя ошибка как (FAR[t]+FRR[t])/2, где t – порог на значение меры близости моделей.

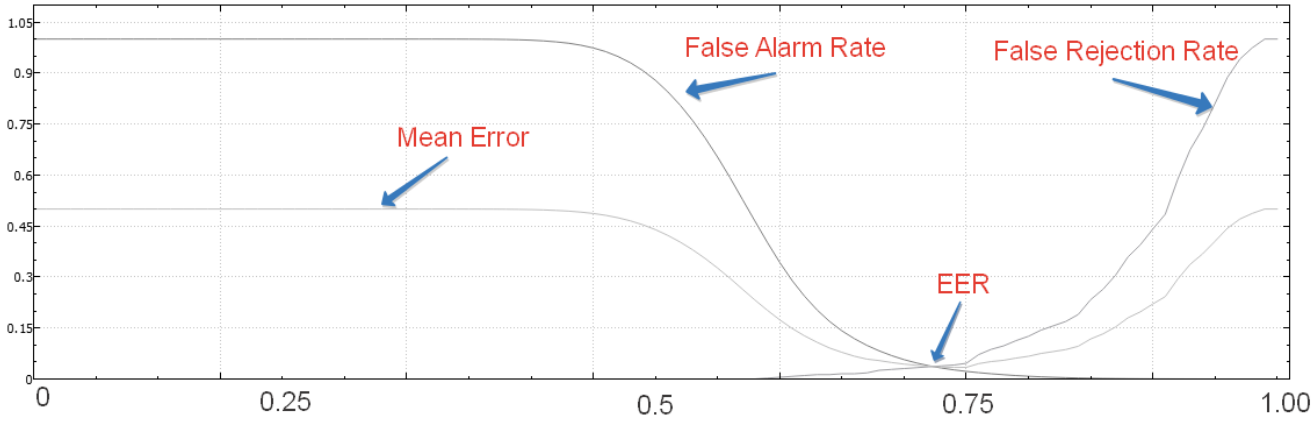

Рисунок 2.1 Графики ошибок первого и второго рода (phone)

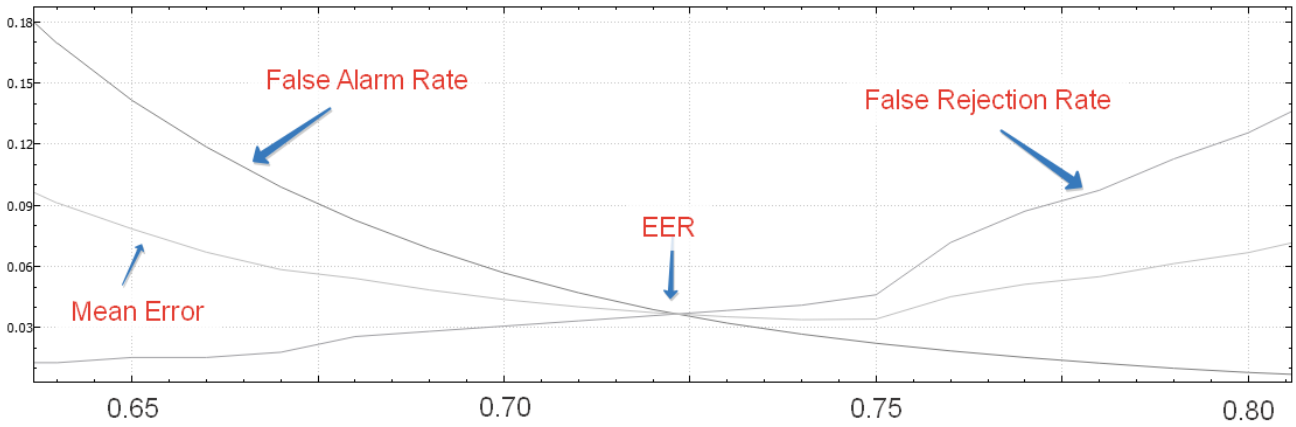

Рисунок 2.2 Графики ошибок первого и второго рода (phone)

4. ВЫЗОВ И ЗАГРУЗКА ПРОГРАММЫ

4.1 Установка 3i Speaker ID SDK

Установка модуля 3i Speaker ID SDK осуществляется переносом (копированием) архивов с модулями с загрузочного диска в требуемую директорию на жестком диске сервера. После переноса (копирования) архива модуля динамические библиотеки (DLL) модуля 3i Speaker ID SDK разархивируются из него в требуемую директорию на жестком диске сервера. Модуль готов к эксплуатации.

4.2 Получение лицензионного файла-ключа

Бинарные файлы модулей 3i Speaker ID SDK защищены от нелицензионного использования и копирования, лицензионное использование предполагает получение файла-ключа, который специфичен для системы пользователя и бинарного файла модуля. Таким образом, полученный файл-ключ нельзя использовать на другой машине или для другого модуля, что исключает нелицензионное копирование и использование модулей. Файл-ключ может быть получен у технической поддержки разработчика в случае наличия у пользователя лицензии. Процедура получения файла-ключа такова:

1) В папке каждого модуля находится директория GenHardID, содержащая исполняемый файл GenHardID_console.exe, генерирующая уникальный идентификационный код системы пользователя. После распаковки архива модуля следует запустить данный файл.
2) Результатом работы GenHardID_console.exe является файл hardware_id.hid, содержащий идентификационный код системы пользователя. Это код одинаков для любого модуля и характеризует систему (машину) пользователя, таким образом, допустимо получить этот файл один раз. Файл hardware_id.hid появляется в той же папке, в которой находится исполняемый файл GenHardID_console.exe.

3) Файл идентификационного кода системы hardware_id.hid следует передать в техническую поддержку 3i, сопровождая информацией о лицензии.
4) При подтверждении пользовательской лицензии, техническая поддержка высылает файл-ключ пользователю. Файл-ключ уникален для каждого исполняемого файла (библиотеки, DLL). Его следует положить рядом с тем бинарным файлом (исполняемым модулем, DLL), для которого файл-ключ был сгенерирован. После этого, модуль будет способен запуститься и работать.

5. ВЫПОЛНЕНИЕ ПРОГРАММЫ

5.1 Комплект модуля 3i Speaker ID SDK

Модуль 3i Speaker ID SDK имеет классический интерфейс, характерный для динамически подгружаемых библиотек (dll). В комплекте с модулем поставляется:

- бинарный файл динамически связываемой библиотеки (расширение dll);
- бинарный файл прокси-библиотеки для статического связывания приложения пользователя с динамически связываемой библиотекой (расширение lib);
- файл-хидер, содержащий исходный код интерфейса модуля на C++ для сборки связывания динамической библиотеки модуля и приложения пользователя (расширение h);
- модель для работы встроенного детектора речи – бинарный файл с нейронной сетью (расширение dnn), конфигурационный файл для вычисления акустических признаков (расширение conf), бинарный файл для их нормирования к среднему уровню (расширение vct);
- модель для идентификации личности по голосу – бинарный файл с нейронной сетью (расширение dnn), конфигурационный файл для вычисления акустических признаков (расширение conf), бинарный файл для их нормирования к среднему уровню (расширение vct);
- файлы с данными для понижения размерности идентификационного вектора (каталог eigens_std): собственные вектора (расширение mtx), собственные числа (расширение vct) , вектор средних значений (расширение vct);
- файлы с данными для процедуры уменьшения межсессионной вариативности (каталог eigens_ch): собственные вектора (расширение mtx), собственные числа (расширение vct) , вектор средних значений (расширение vct).

5.2 Описание функционала API

Список функций API 3i Speaker ID SDK представлен в таблице 5.1.

Таблица 5.1 Список функций API 3i Speaker ID SDK

Наименование функции	**Описание функции**
sid_init	Инициализация библиотеки
sid_deinit	Деинициализация библиотеки
sid_is_init	Проверка статуса инициализации библиотеки
sid_voice_model_calc_from_buf *sid_voice_model_calc_from_file*	Построение векторной модели голоса диктора. ***Источник:*** буфер памяти или WAV-файл
sid_voice_model_save	Сохранение векторной модели голоса диктора в бинарный файл
sid_voice_model_load	Загрузка векторной модели голоса диктора из бинарного файла
sid_voice_models_compare	Вычисление меры близости между двумя моделями голосов (непосредственно идентификация)
sid_voice_model_destroy	Очистить и удалить контейнер с моделью
sid_version	Информация о версии библиотеки
sid_get_err_msg	Получение текстового описания кода ошибки

Описание функций API 3i Speaker ID SDK представлено ниже.

Дополнительные типы данных

```
#define voice_model_base_t    double
#define voice_model_base_ptr  voice_model_base_t*

struct voice_model_t
{
  voice_model_t(int modelSize = 0,
                voice_model_base_ptr modelData = nullptr,
                int signalDur = 0,
                int speechDur = 0);
```

```
  ~voice_model_t();

  int SpeechDuration; // Количество речи, на которой вычислена
                      // модель (мсек)
  int SignalDuration; // Общая длительность аудио записи(мсек)
  int ModelSize;      // Размер вектор-модели
  voice_model_base_ptr Model; // Данные вектор-модели
};
typedef voice_model_t* voice_model_ptr;
```

Инициализация библиотеки

Прежде, чем приступить к работе с библиотекой, необходимо ее инициализировать. В процессе инициализации производится загрузка всех нейросетевых моделей, данных для проекции векторов на главные компоненты (PCA – Principal Component Analysis), а также параметров вычисления акустических признаков, поставляемых вместе с модулем (см. п.5.1).

API функции:

```
int sid_init(char const* model_dir_path);
```

model_dir_path – путь к каталогу с файлами, необходимыми для обнаружения речи и вычисления модели голоса диктора.

Не допускается изменение структуры каталога, содержащего файлы модел.

Функция возвращает 0 в случае успеха или код ошибки.

Деинициализация библиотеки

По окончании работы с библиотекой необходимо высвободить все ресурсы, занятые при инициализации библиотеки.

API функции:

```
void sid_deinit();
```

После вызова данной функции будет заблокирована возможность работы с функциями:

sid_voice_model_calc_from_file(),sid_voice_model_calc_from_buf(), sid_voice_model_save(), sid_voice_model_destroy(), sid_voice_models_compare().

Функция *sid_deinit()* будет «ждать» окончания работы всех запущенных ранее функций из числа перечисленных, а по завершении их работы выполнит освобождение занятых ресурсов. Любая из этих функций, вызванная в момент «ожидания», завершится кодом IS_BLOCKED=8.

Если на момент начала высвобождения ресурсов не все голосовые модели удалены, это будет выполнено функцией *sid_deinit()*.

Функция возвращает код согласно таблице возвращаемых кодов.

Проверка статуса деинициализации библиотеки

API функции:

```
bool sid_is_init();
```

Функция возвращает следующий результат:

TRUE – библиотека успешно проинициализирована и готова к работе с аудио данными;

FALSE – библиотека не проинициализирована или деинициализирована, или находится в состоянии деинициализации.

Получение информации о версии библиотеки

API функции:

```
char const* sid_version();
```

Функция возвращает указатель на строку вида <MAJOR_VER>.<MINOR_VER>.<PATCH_VER>.

Построение модели голоса диктора («базовые» функции)

API функции (источник - WAV-файл):

```
voice_model_ptr sid_voice_model_calc_from_file(int& res_code,
char const* wav_fpath);
```

res_code - адрес для записи кода завершения операции согласно табл. 7.1;

wav_fpath - путь к WAV-файлу.

API функции (источник – буфер памяти):

```
voice_model_ptr sid_voice_model_calc_from_buf(int & res_code,
short const* audio_data_ptr, int audio_data_size);
```

res_code - адрес для записи кода завершения операции согласно табл. 7.1;

audio_data_ptr - указатель на массив отсчетов;

audio_data_size - количество отсчетов в аудио буфере.

При обработке звуковых данных большого объёма стоит учитывать, что векторная модель голоса строится по аудиоданным первых 5 секунд речи, обнаруженной во входном сигнале.

В случаях, когда во входном сигнале присутствует речь более, чем одного диктора, модель голоса будет «некачественной», поскольку будет в определённой мере соответствовать каждому голосу.

Чтобы обеспечить оптимальный расход памяти и минимизировать время построения модели, рекомендуется делить входной аудио сигнал на части длительностью не более 1 минуты.

Обе функции являются потокобезопасными, а значит, что они могут быть вызваны из нескольких потоков одновременно.

В случае успеха каждая функция возвращает указатель на вектор-модель голоса диктора, а в *res_code* записывает код 0, иначе – возвращает nullptr, а в *res_code* пишет код ошибки.

Сравнение векторных моделей голосов

API функции:

```
int sid_voice_models_compare(float & confidence, const
voice_model_ptr target_voice_model, const voice_model_ptr
unknown_voice_model);
```

confidence - адрес для записи значения достоверности результата идентификации (сравнения двух голосов);

target_voice_model – указатель на векторную модель «целевого» (известного) голоса;

unknowm_voice_model – указатель на векторную модель «неизвестного» голоса.

В переменную *confidence* будет записана достоверность того, что идентифицируемый диктор является «целевым». Значение достоверности имеет диапазон [-1;1]. Как видно на рисунке 2.2 равновероятная ошибка достигается при пороговом значении на достоверность равном 0.72. Для получения более достоверного результата рекомендуется принимать результаты, достоверность которых выше 0.72. Стоит учитывать, что в таком случае возрастает вероятность пропуска «целевого» диктора. При пороге достоверности ниже 0.72 увеличивается вероятность приёма «чужого» диктора за «целевого».

Сравниваемые модели должны быть построены с использованием одинакового набора моделей.

Функция возвращает 0 в случае успеха, иначе - код ошибки.

Очистка и удаление выходной структуры результатов

API функции:

```
void sid_voice_model_destroy(voice_model_ptr & voice_model);
```

voice_model - указатель на векторную модель голоса, которая возвращается функцией *sid_voice_model_calc_from_buf(), sid_voice_model_calc_from_file(), sid_voice_model_load()*.

Получение строки с текстом ошибки

API функции:

```
const char* sid_get_err_msg(int err_code);
```

err_code - код завершения операции, возвращаемый функциями библиотеки.

Функция возвращает указатель на строку с текстовой интерпретацией кода ошибки.

Сохранение векторной модели голоса в бинарный файл

API функции:

```
int sid_voice_model_save(const voice_model_ptr voice_model, char const* dst_fpath);
```

voice_model - ссылка на сохраняемую векторную модель;

dst_fpath - путь к файлу, в который будет сохранена модель.

Функция возвращает код согласно таблице возвращаемых кодов.

Загрузка векторной модели голоса из бинарного файла

API функции:

```
voice_model_ptr sid_voice_model_load(int& res_code, char const* src_fpath);
```

res_code - адрес для записи завершения работы функции;

src_fpath - путь к файлу с моделью.

В случае успеха функция возвращает ссылку на загруженную векторную модель, а в *res_code* записывает код 0, иначе - возвращает nullptr, а в *res_code* будет записан код ошибки (см. Таблицу 7.1).

6. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Требования к входным аудио данным 3i Speaker ID SDK:

- возможные источники: WAV-файлы, буфер отсчётов;
- частота дискретизации сигнала: 8 кГц;
- разрядность квантования: 16-бит;
- тип кодирования, если источником является WAV-файл: A-закон, Mu-закон или PCM (без дополнительных блоков типа list или fact);
- тип кодирования, если источником является буфер памяти: PCM.

Требования к качеству сигнала:

- значение ОСШ должно составлять не менее 10дБ;
- допускается присутствие посторонних звуков в виде однотональных гудков и сигналов тонального набора, DTMF.

Выходные данные: число с плавающей точкой confidence, отражающее достоверность того, что идентифицируемый диктор является «целевым». Значение достоверности имеет диапазон [-1;1].

Равновероятная ошибка между ошибками первого и второго рода достигается при пороговом значении на достоверность равном 0.72.

7. СООБЩЕНИЯ ОПЕРАТОРУ

Все сообщения оператору реализуются через коды возвращаемых значений функциями модуля, далее дается расшифровка (см. Таблицу 7.1).

Таблица 7.1 Коды возвращаемых значений функциями модуля

Код	**Текстовая интерпретация**	**Код**	**Текстовая интерпретация**
0	NO_ERR	**11**	INITIALIZATION_ERROR
1	UNKNOWN_ERROR	**12**	DEINITIALIZATION_ERROR
3	MEMORY_ALLOCATION_ERROR	**27**	INSUFFICIENT_SIZE_OF_THE_SOUND_BUFFER
4	INCORRECT_POINTER	**29**	UNABLE_TO_SAVE_FILE
5	INCORRECT_INPUT_PARAM	**30**	UNABLE_TO_OPEN_FILE
6	INCORRECT_INPUT_CONTAINER	**32**	INCORRECT_FILE_FORMAT
7	INCORRECT_INIT_PARAMS	**35**	UNABLE_TO_GET_FEATURES
8	IS_BLOCKED	**41**	UNABLE_TO_LOAD_MODEL
9	IS_NOT_INIT	**44**	UNABLE_TO_LOAD_PCA_DATA
10	ALREADY_INIT	**54**	UNABLE_TO_GET_IVECTOR

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

В настоящем документе приняты следующие условные обозначения:

СПО	Специальное программное обеспечение
СУБД	Система управления базами данных
3i Speaker ID SDK	СПО «3i Speaker ID SDK »
API	сокр. англ. Application Programming Interface, интерфейс программирования приложений – набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением для использования во внешних программных продуктах
DLL	сокр. англ. Dynamic Link Library – динамическая библиотека, позволяющая многократное использование различными программными приложениями.
CPU	сокр. англ. Central Processing Unit – электронный блок либо интегральная схема (микропроцессор), исполняющая машинные инструкции (код программ).
DNN	сокр. англ. Deep Neural Network, искусственная нейронная сеть с несколькими скрытыми слоями.
WFST	сокр. англ. Weighted Finite State Transducer, взвешенный конечно-автоматный преобразователь, автомат Мили со взвешенными дугами.
SDK	сокр. англ. Source Development Kit – комплект средств разработки, который позволяет специалистам по программному обеспечению создавать приложения.

GMM	сокр. англ. Gaussian Mixture Model – статистическая модель плотности вероятности, выраженная суммой нормальных многомерных распределений.
MFCC	сокр. англ. Mel-Frequency Cepstrum Coefficients – представление кратковременного спектра, основанное на линейном косинусном преобразовании логарифмированного амплитудного спектра по мел-частотной шкале.
DTMF	сокр. англ. Dual-Tone Multi-Frequency, двухтональный многочастотный аналоговый сигнал, используемый для набора телефонного номера.
UBM	сокр. англ. Universal Background Model – статистическая модель пространства признаков голоса.
VST	сокр. англ. Voice Segmentation Technology – технология сегментации по голосам
PCA	сокр. англ. Principal Component Analysis – метод сокращения размерности статистических данных.
PCM	сокр. англ. Pulse Code Modulation, импульсно-кодовая модуляция, термин применяется в смысле типа кодирования аудио-сигнала.

Лист регистрации изменений

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа и дата	Подпись	Дата
	Измененных	замененных	новых	аннулированных					