Scribo

Calin Buruian: calinburuian@gmail.com
Isa Usmanov: cordell.box@gmail.com
Janis Postels: j.postels@gmx.de
Piotr Tatarczyk: tatarczykp@gmail.com
Vladimir Fomenko: vlfomenk@gmail.com

Scribo

A revolutionary AI-enabled offline audio-to-text transcribing mobile application.

Introduction

For the TechChallenge the Scribo team was tackling the HUAWEI challenge of leveraging their NPU chip to build AI-powered mobile applications. Our diverse team that consisted of software engineers, data scientists and business enthusiasts have successfully delivered an MVP in the form of the mobile app.

Problem

Problem definition

We were tackling the problem of people who need to record long audio recordings for their professional needs, i.e. journalists. Journalists, taking ~8 interviews per month which are, on average, 30 minutes long, then must transcribe their recordings. That takes them up to 4 times longer than the time of the original recording.

It appears that it is very crucial for the journalists to obey legal and also corporate requirements which do not allow them to share the recordings before publishing the article. It means that they often cannot let somebody else transcribe their recording or use apps that do transcription in the cloud.

Our app solves this problem.

Solution

Cooperating with HUAWEI, we developed Scribo - the first AI-enabled on device transcribing Android application. HUAWEI's futuristic Kirin 970 NPU allowed us to develop a technology that can transcribe the audio recording without using any internet connection, which means delivering results without using any of the available cloud solutions.

The audio is recorded on the phone, processed on the phone and edited on the phone. We are pioneering in ondevice transcribing technologies. Our solution can be used

without any fear of somebody stealing/misusing your data as you are the only possessor of your recording.

In this document we briefly describe the process of the implementation of our application. We published our code for the reference.

Technical description

Application development environment

The Scribo mobile app is built using an official android development platform - Android Studio 3.2.1 with the following Java Runtime Environment: 1.8.0_152-release-1136-b06 x86_64.

For testing purposes the HUAWEI's P20 mobile phone was used, therefore the target SDK version of the development project is 28 with a minimum SDK version being 27.

Application structure

Our app consists of 3 main views: 1) *MainActivity*, 2) *RecordingListActivity*, and 3) *RecordingDetailsActivity*. (See the screenshots at the end of this document)

MainActivity is the view where the audio is recorded. Every audio is recorded in the .wav format. Android studio does not have an out-of-the-box solution for recording audio in the .wav format, so we built our recording based on this [Link 1] open source solution.

The "recording" button and its animation is implemented based on the opensource solution provided by Yang Hui [Link 2]. When the recording is stopped and the "save" button is pressed the user is asked to give a name to the audio recording. Then the information of the recording (time, date, duration, name, id, uri) is stored into the local database using the Android's built-in solution - The Room Persistent Database [Link 3]. Finally, the "list" button triggers to open another activity - the *RecordingListActivity*.

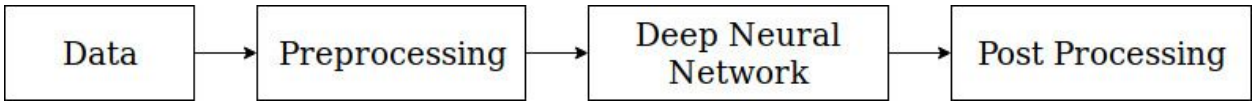
The *RecordingListActivity* loads the local database of the recordings and displays the recording in the adapted list view. Rows are modified with a customized list-view adapter to display title, date and duration of the recording and also shows the status of the recording whether it is already transcribed or not. The "transcribe" button launches the transcribing process. After the transcribing is performed (transcribing and machine learning parts are explained in detail later) we store the transcription of the recording into the database locally on the phone. Then when user presses on the list item the *RecordingDetailsActivity will open.*

RecordingDetailsActivity is one of the most difficult parts in terms of UI implementation. To play the audio and to have the played word highlighted on the text view, we associated every word in the transcription with a timestamp. You can find how we implemented in detail the “karaoke” feature on our GitHub repository [Link 4]. Additionally, we implemented an intelligent text editor that would propose a likely replacement of the wrongly transcribed word. Each word is associated with an integer. And if you click on the wrong word it would suggest you the nearest neighbouring words. For a very simplified example: *like (#45)* suggestions => hike (#44), bike(#46), *liked (#47).* Implementation of the word suggesting can also be found in the code.

Last but not least, the Scribo app allows quickly to share the transcription material with anyone via email. By pressing the “Share” button the app will trigger native mailing app to open and will fill in the content of the message with the transcribed text.

Speech recognition system

Overview



The speech recognition system is comprised of four components (see above). The data is being loaded from the dataset which consists of short audio files and the corresponding transcriptions. During the subsequent preprocessing step we extract MFCC features from the audio files. A deep neural network then predicts softmax score (of dimensionality 28 - one entry for every character used in training) for every time step. In the post processing stage we apply beam search algorithm to find the optimal sequence of characters which can be of variable length. Beam search algorithm is applied in combination with a language model (for details check the beam search/language model section) that ensures that the predicted words are drawn from a predefined vocabulary of allowed words.

In the following we describe all the stages in greater detail and give sufficient detailed information to reproduce the training process.

Reasons for training our own model

Especially for english language there are various pretrained model of moderate quality readily available (see https://github.com/mozilla/DeepSpeech, https://github.com/buriburisuri/speech-to-text-wavenet). Unfortunately we found throughout the project that all of those models contained a considerable amount of operations that are not compatible with Huawei NPU. This way we identified the need for implementing our own custom solution.

Data

For the training of our final deep neural network we use four publicly available speech datasets for english language:

- LibriSpeech: 1000 hours of transcribed speech derived audio books
- Tatoeba: ~600 spoken short example sentences with transcription aimed for studying english as a foreign language
- VoxForge: Several short speech segments from 6240 speakers. Collected via crowdsourcing approach
- TEDLIUM-releasev3: 452 hours of transcribed speech from TED talks. Each talk is split up into small audio segments with corresponding transcription.

Combining all four datasets we training on roughly 2000 hours of speech data.

Data Preprocessing

The data preprocessing takes up a large part of the actual workload for several reasons:

- Datasets are in different formats and need to be unified before merging.
- Due to the size of the dataset, the data does not fit into RAM. Thus special thought has to be put into loading data efficiently from disk.

- Since we aim at compatibility with the phone while training using python in a linux environment, all transformations of the data from recorded audio to input tot the neural network need to be exactly reproducible in Java or C++

Before merging the datasets we transform all underlying audio files to wav-format with a sampling rate of 16kHz and split the data into training set (80%) and validation set (20%). This is also the format of our recording in the ScriboAI app. For training purpose (since we have to use static LSTMs on Huawei NPU) we need all input data of the same length. We examined the length distribution of the audio segments in the overall dataset and decided to use a maximum length of 15.36 s. The decimal part of the maximum duration is due to the fact the Huawei NPU just supports sequence lengths which are multiple of 16 and this duration yields a sequence of MFCC features which is 960 long. All audio clips longer than that are discarded. We computed MFCC features on these audio files. For computing features we use the Python library Librosa with the following configuration: (sample_rate=16000, n_fft=2048, hop_length=256, n_mfcc=16, n_mels=128, fmin=0, fmax=8000).

We translated the functionality of above library into Java code for our application. For a 15.36 s audio segment this yields exactly 960x16 dimensional MFCC features. We normalize those features individually for each sample by subtracting the mean and dividing by the standard deviation. This yields comparable results and eliminates the need for recomputing the mean and variance for the entire dataset every time we extend the entire dataset with new data.

The text target is being transformed into sequences of integers where each each integer represents one specific character.

For sequences that are shorter than the maximum duration we pad the data to a length of 960 (MFCC features with 0s and the targets with a dedicated integer representing the end of the text)

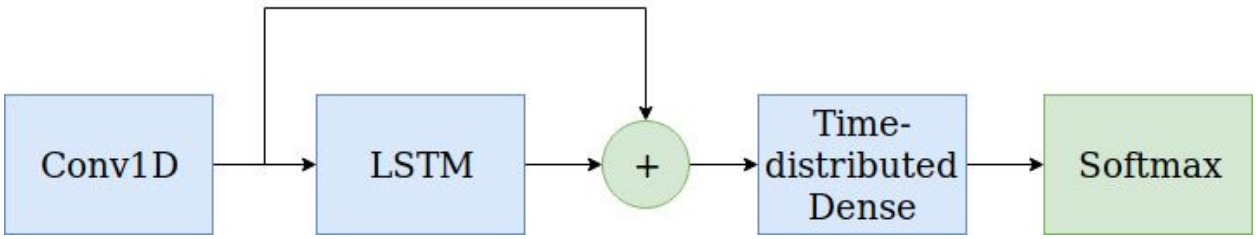
In order to load the data efficiently during training we randomly assemble mini batches of size 32x960x16 and save them in .npy-format which is optimized for numpy. During

the final training we train a batch size of 64. Therefore we randomly select two mini batches and feed them to the network.

Since we were unable to use 1D convolutions (for details see next section) we rearange the data into 64x480x160 to simulate a 1D convolution with a time distributed dense layer.

Deep Neural Network (ScriboNet)

We developed our own custom architecture for this challenge (see below graphic).



The basic building blocks that we were able to successfully import on the phone were:

- Conv1D: We could not directly import a 1D conv layer. But we “simulated” it by reordering the data in the preprocessing step and applying a time-distributed dense layer. This way were unfortunately just able to deploy it once in the begging of the network.
- Static LSTM: We were able to deploy static lstm layer to the phone. We were not able to stack several LSTM layers as this led to DevEco getting stuck during import.
- Time-Distributed Dense: We were able to deploy one time distributed dense layer in a row (stacking them led to the same problem as for the lstm).
- Residual connection: We found empirically that using a residual connection bypassing the LSTM boosts performance. It is motivated by findings in computer vision where learning the deviation from the input is easier that learning the entire nonlinear mapping with a layer. It further benefits gradient flow to the first layer.
- Softmax: We use softmax during training but we did not embed it into our model since Huawei mentioned that this is not supported. So, on the mobile device we

reimplemented softmax to compute the final scores from the last time-distributed layer.

We use beam search with subsequent CTC loss (standard tensorflow implementation) for performing backpropagation during training

In the following we name details with respect to hyperparameter settings:

- Input data:
  - Shape batch_sizex480x160
- Conv1D:
  - Kernel size = 10
  - Stride = 2
  - Number kernels = 1
  - Output shape = batch_sizex480x256
- LSTM (important for Huawei NPU → all numbers must be divisible by 16):
  - Hidden dimension = 256
  - Time steps: 480
  - Activation: tanh
  - Output shape: batch_sizex480x256
- Time distributed dense:
  - Output shape: batch_sizex480x32 → 32 because a concatenation follows this layer and it is also required by Huawei the have a output dimension divisible by 16
- Learning rate: initially 0.003. When hitting a plateau in validation loss the learning rate is reduced by a factor of 0.75. Minimum learning rate is 0.00001
- Patience: we stop training when we did not observe improvement in validation loss for 6 epochs in row.

The detailed code and the pretrained model are available under: https://github.com/janisgp/SpeechRecognitionHuaweiNPU

Beam search/Language model

To recap, given an input audio of length T_audio seconds, we split it into chunks of 15.53 seconds that are transformed into arrays of MFCC features of shape (960, 16). For every chunk the SpeechRecognition model outputs a sequence of shape (480, 28) that corresponds to the probabilities of 28 characters (27 english letters + space) being pronounced on every of 480 frames (~32ms per frame). We firstly concatenate the outputs for every chunk and obtain a sequence of shape (480 * [T_audio/15.53s] = T_sequence, 28) which corresponds to the probabilities of characters said for the whole recording and we work with it in the following steps.

We will be using a **Beam search**, that is just a heuristic search algorithm that explores a solution by expanding the most promising paths in a limited set. In our case, we have a set of all possible sequence characters of length T_sequence (that we are going to limit soon) and our goal is to find the most likely sequence of characters based on the given character probabilities for every frame (node).

The **most primitive approach** to solve this problem would be just to find a sequence of characters that has the highest joint probability: $max(\prod_{1}^{T_{sequence}} c_i)$. Taking into account that all the probabilities are independent, we would get that the solution to this problem is just: $\prod_{1}^{T_{sequence}} max(c_i)$, so for every frame we can just select the most likely letter to has been said in that frame. However, following the current approach we might end up having the sequences that have no meaning in English such as: “heealooo, hoaow aar uu”, which is likely just “hello, how are you”. As you might guess, we can improve our model a bit by introducing some easy tricks such as merging the sequences of identical characters, but still it would not be enough to get a decent result.

To improve the results, we introduce a new model that is going to adjust the probabilities of every character for the Beam Search, **the Language Model**.

The simplest Language Model (that we also started with) that we can add would just limit all the possible constructed words to the **English vocabulary**. So now, whenever the Beam Search tries to add some letter C to the existing sequence S, we should firstly check whether there are any available words with the prefix S+C in the vocabulary available (e.g. having prefix “sk” we can’t add letter “z” to the sequence even if SpeechRecognition model said it’s the most probable one, because “skz” is not a valid prefix for any English word).

However, by introducing such limitations the solution to the construction of the most likely sequence task is not just the product of most likely characters anymore, and **the solution to the new task becomes NP-hard**. The reason is that for the current frame we can’t be sure if picking the most likely character is the best strategy - in the next step we might not be able to pick the most likely character again because of no words that will match the obtained prefix. This means that now sometimes picking a character that has not the highest probability makes sense.

In order to tackle this new problem, we can just try to build all the possible sequences, e.g. pick every character for every frame (ensuring that the formed words are present in the vocabulary) and then just pick the most likely sequence in the end. However, this would require a memory and time complexity of O(28^T_sequence), which is, of course, too much. To solve this challenge, we can **limit the capacity of sequences set at every iteration**, so after picking all possible characters for the current frame, we select top N most likely sequences and leave out the rest. Empirically we have found that this approach gives us plausible results with N being at least 10 but we did not find the perfect value (there is always a trade-off between how much you trust the SpeechRecognition model and how much you trust Language Model statistics; if the speech model is not good enough, you might want to set your N to the lowest possible value).

We, however, further improved our Language Model by **introducing precalculated Bigram probabilities** to the Beam Search as an additional multiplier at every step.

Specifically, before introducing this, the probability of the sequence after the adding of character c was defined as:

$$p_{sequence[i]} = p_{sequence[i-1]} \cdot c_i \cdot [0 \text{ if there is no words with prefix "} sequence[i-1] + c_i \text{"}]$$

and with taking bigram probabilities into account it now becomes:

$$p_{sequence[i]} = p_{sequence[i-1]} \cdot c_i * p_{bigram}[word_{previous}][word_{current\ prefix + c_i}],$$

where the last multiplier is the sum of the probabilities of that the previous word is followed by some word with the prefix [current_prefix + c]. This basically means that during the selection of the next character we consider how likely is that there exists a word that can be built in the future that follows the previous word that we already selected before.

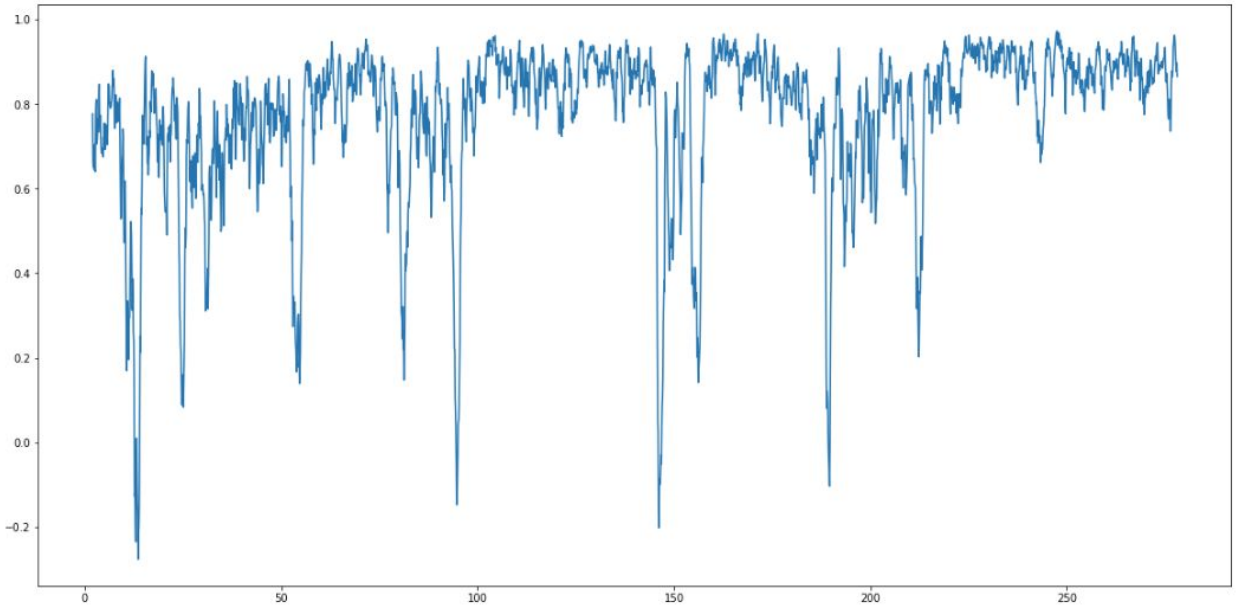
To calculate the bigram probabilities we used **English corpus from OpenSubtitles dataset** (http://opus.nlpl.eu/OpenSubtitles2016.php).

Speaker change detection

Text post-processing greatly influences the user experience in speech recognition tools. When working with a recorded interview, one deals often with texts of thousands of words. Searching, scrolling and finding information get you a tedious job. Text separation according to speakers is a means for solving those problems. Our speaker change detection is based on the embedding neural network trained for speaker identification. The training process and architecture were thoroughly described in the attachment - “Speaker Identification Challenge” and those part will be skipped here.

For detection of speaker turns, MFCC features were used. To reduce the overall computational cost, the same parameters were used as for the speech recognition. The input to the embedding network are two neighbouring “windows” of speech, both 2.24 sec long (eg. from 0 to 2.24s and 2.24s to 4.48s). With the parameters used, it corresponds to 224 feature vectors per window. Embeddings of the two windows are computed, and subsequently the cosine similarity score between them. In the next step,

we shift both windows in time by 0.1s and repeat the process. The outcome is a plot of distance between embeddings with respect to time. Once a threshold is found, the similarities below this value indicate a change of speakers. We set a maximum of one speaker change per second. Detection of changes is not possible within the first and last 2.24 sec of the audio.



Further recommendations: One can try to cluster speakers based on the predicted speech segments and changes. This task is often called as speaker diarization, and deals with “who speaks when”. To increase the accuracy, speech detection can be used.

Speaker identification

See separate attachment - “Speaker Identification Challenge”

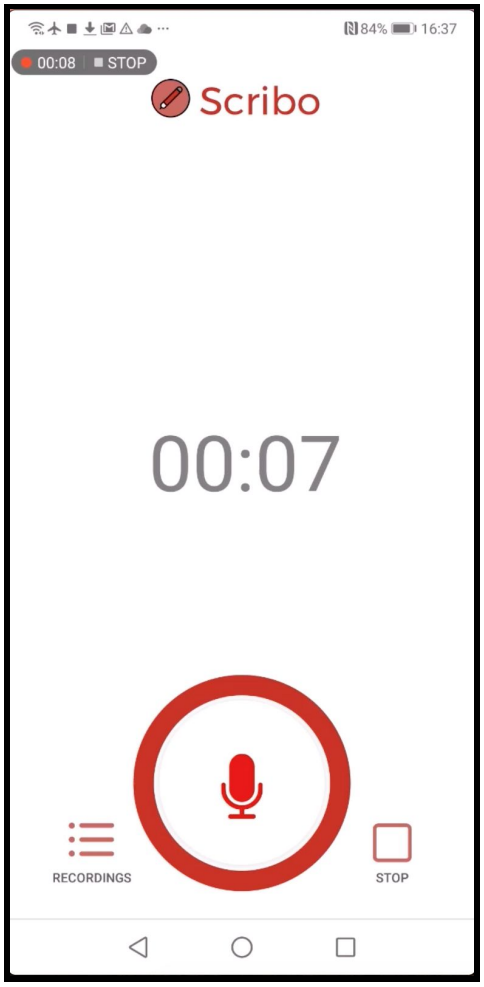
Summary

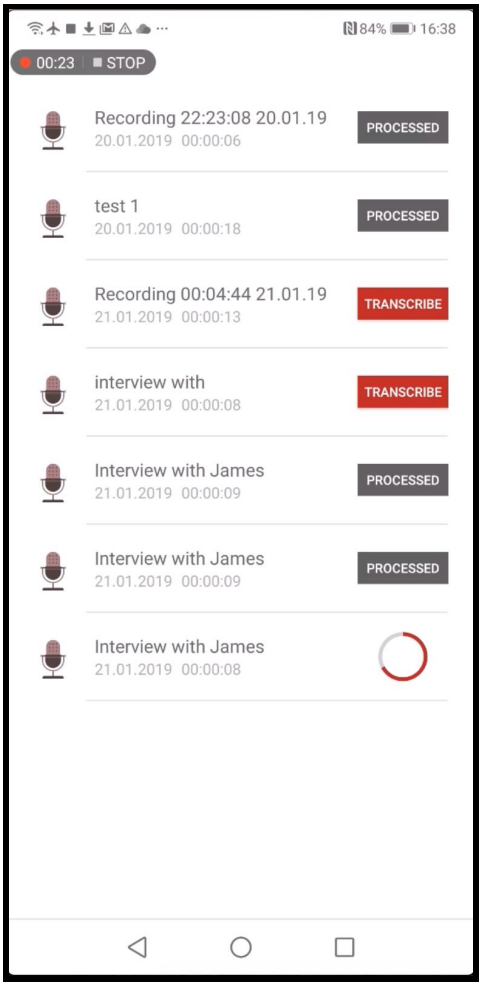
During the TechChallenge the Scribo app has been developed to a decent state. We are motivated to proceed with developing it further and hope to cooperate with

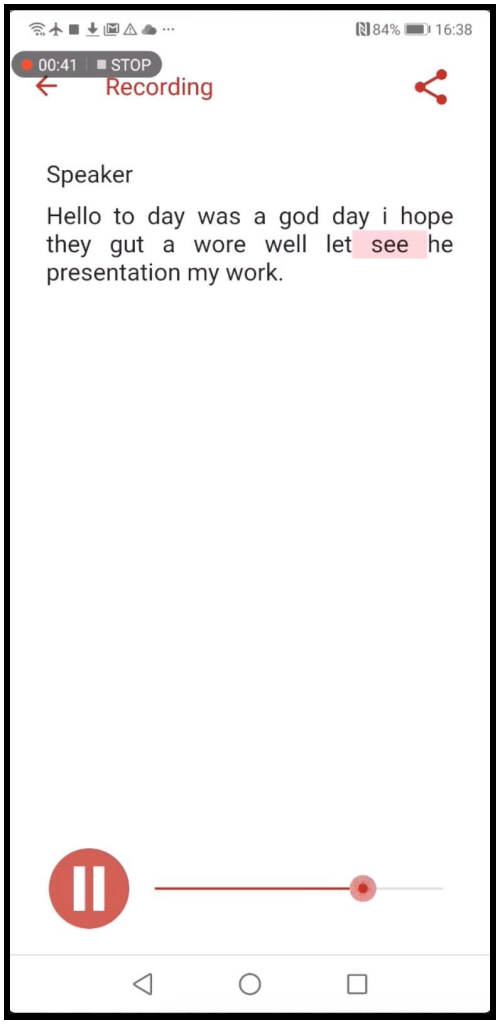
HUAWEI’s developers in order to implement the most secure on the market transcribing application.

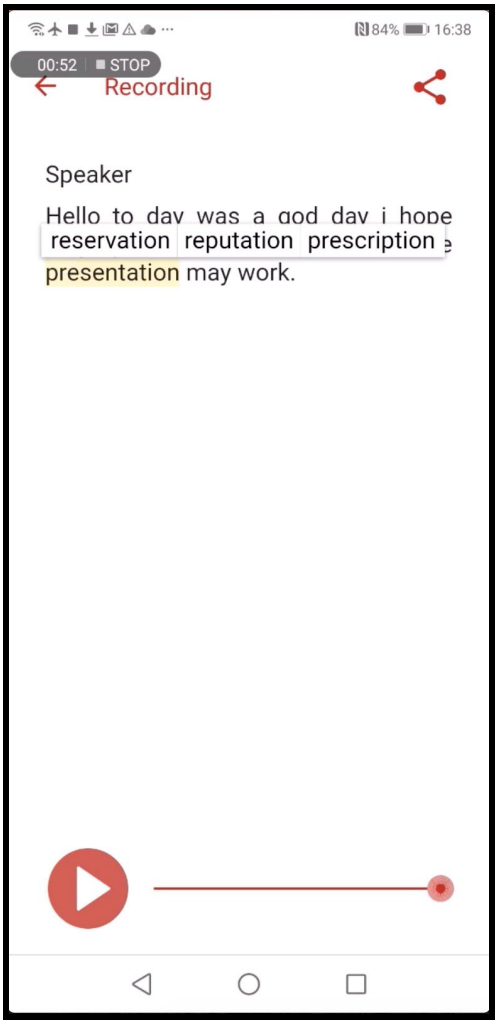
Screenshots

Below you can find the screenshots of the main screen of the app during the launch, during the recording, the screen with list of recordings and the screen with the transcription results.









Code

[Link 4]: The code is available publicly on the github.

Links

[Link 1]: Selvaline's blog - Record audio wav format Android [How to - Guide - Tutorial]: http://selvaline.blogspot.com/2016/04/record-audio-wav-format-android-how-to.html

[Link 2]: GitHub of Yang Hui.

[Link 3]: Official Android Documentation - The Room Persistent Database