Speech Recognition for Mobile Linux Distrubitions in the Case of Aurora OS

Alexey Andreev
Open Mobile Platform
St.Petersburg, Russia
a.andreev@omprussia.ru

Kirill Chuvilin
Open Mobile Platform
Moscow, Russia
k.chuvilin@omprussia.ru
Moscow Institute of Physics and Technology
Moscow, Russia
kirill.chuvilin@phystech.edu

***Abstract*—Aurora OS is a POSIX-compatible mobile Linux distribution designed for enterprise and business purposes. The main application area of devices based on Aurora OS is corporations in Russia. For example, such devices are used by field workers to receive tasks and send reports. The reports are generated by filling out complex forms with a large number of input fields and options to choose from. Speech recognition allows to significantly speed up this process. Aurora OS has no built-in tools to implement such functions, so developers of software for field workers need to use a third-party solution. There are several voice recognition engines for POSIX-compatible systems, but only those that support the Russian language, can work locally on devices, and have a free license are suitable for the task. The only solution for the task that meets the criteria is the Kaldi engine. However, it depends on libraries implemented using Fortran, and Aurora OS does not support this programming language. Therefore, it was necessary to develop a way to use software implemented in Fortran in an environment that does not have a native support for it. This paper proposes an approach for solving such a problem, which can be applied to all similar cases.**

I. Introduction

A. Speech recognition on mobile devices

Automatic speech recognition (ASR) is convenient for many aspects of mobile device use:

- switch the modes;
- setup the alarms volume and notification modes;
- launch available applications;
- control connected devices and sensors;
- call contacts.

For mobile operating systems aimed at the consumer market, such solutions come by default: Siri for iOS and iPadOS and Google Assistant for Android [1], [2]. They provide both solutions for system UI and APIs for third-party developers to interact with them.

The speech recognition feature (*speech to text*, STT) is also in wide demand. It allows you to significantly speed up text input, especially in cases where the use of other input methods is difficult or limited. This is useful, for example, for the following tasks:

- notes taking;
- composing of shorts messages;
- speech translation.

An example of such input solution is GBoard keyboard application [3].

There are two approaches to ASR implementation: client-server or local solution. In some cases, mixed approaches are used. Client-server solution is often easier to implement. In addition, the server provides more computing power than the local device, which allows both speeding up recognition and improving its quality. For example, Apple provides Speech framework and Google provides Speech-to-Text Cloud. However, modern mobile devices are already able to recognize a large number of voice commands or even to natural speech in general. The recent updates from the Android and iOS also introduce offline speech recognition on a device [4], [5].

In addition, it is often unacceptable in business tasks to transfer sensitive information to the server.

B. The case of Aurora OS

Aurora OS is a POSIX-compatible Linux distribution designed to be used on mobile devices in the B2B and the B2G segments [6]. Several hundred thousand Aurora-based mobile devices are now used in commercial projects with corporations in Russia. There is a large number of tasks and situations in which Aurora OS is a convenient solution. The main cases are as follows:

- Employees working in the field use mobile devices to receive tasks, prepare and send reports.
- Executives and line managers of organizations use mobile devices to solve business tasks, including work with the organizer, e-mail and applications for interaction with information systems.

The examples of voice input discussed above are also suitable for each of these cases. But for field workers, there are a few additional tasks in which voice recognition can help considerably:

- checking of field objects status;
- reports preparing.

A common situation for field workers is as follows. They need to fill out a complex form with many fields and options to select from. As a rule, this is due to the large number of characteristics of the objects under study. The touch input in

such tasks could be noticeably slow or difficult comparing to the voice input.

For example, similar voice navigation could also be used in Smart TV voice interfaces. When a user of a Smart TV does not have a full-featured keyboard and only a joystick of the remote, selecting the nested menus with many selection options could be very slow comparing to the voice commands, where the user don't have to manually scroll and switch all the lists of possible options.

Returning to the field workers, the temperature and the weather conditions, in general, could also be crucial in terms of the user experience. The text input via dictation could significantly improve the mobile user experience. So the voice input could be preferable not only in terms of the filling of the predefined forms but also in recognizing the texts of the natural speech, voice.

Solutions for field workers are developed by specialized companies. And often the statement of work includes support for all available versions of Aurora OS, including those released several years ago. The vendor (Open Mobile Platform LLC) provides support for old OS versions related only to fixing critical problems and bugs, but not to expanding functionality or embedding new libraries in the toolchain. This article discusses the problem of implementing a third-party solution for Aurora OS that supports automatic speech recognition features.

Android and iOS solutions cannot be directly reused due to Aorora OS is a POSIX-compatible Linux distribution so low-level subroutines and libraries are incompatible. At the same time Aurora OS differs from other mobile Linux distributions in the inaccessibility of some build tools. For example, there is no Fortran source code build capability in the standard toolchains [7]. In the case of the speech recognition systems discussed in this article, this is quite an important limitation. It is also important to keep in mind that Aurora OS runs on mobile devices, which do not have as extensive computing resources as PCs or server solutions.

C. Solution design

Before discussing available solutions and formulating requirements, the interaction between the application, the speech recognition system and the API provided by the operating system should be described (Fig. 1).

Aurora OS provides Qt as the main framework for third-party software. Therefore, the QtMultimedia module (QAudioRecorder, QAudioDecoder, QAudioProbe classes) is used to capture sound [8]. This way it is possible to produce a sound file or stream. Since the Aurora OS does not include a speech recognition system, it must be supplied by the application developer. Therefore, the application is responsible for transferring the audio stream to the ASR system and processing the result.

II. Speech recognition system requirements

Before considering the available solutions and possible approaches to the implementation of speech recognition, we will formulate the requirements that must be met for the result to satisfy the needs of the applied tasks.

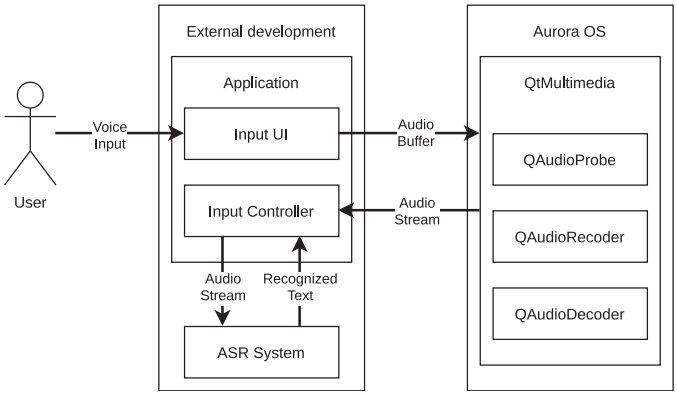

Fig. 1. Solution design

A. Aurora OS support

The initial task is to implement speech recognition specifically for Aurora OS. Aurora SDK is used in software development for this operating system. However, right now it does not provide all the possible tools to build software, so we assume that an additional toolchain can be used to build the necessary components with the possibility to make the corresponding changes in Aurora SDK later.

However, this requirement means that the resulting solution and all necessary components must be able to run on current devices controlled by the Aurora OS.

A common requirement for commercial software projects is to support all the active OS versions, including those already in use in projects. Due to contractual obligations, the vendor (Open Mobile Platform LLC) provides only security updates and critical fixes for the older releases. Therefore, significant changes in the existing environment or the toolchain could not be expected to meet the requirements of any external development. These kinds of updates can only be released with new versions of the OS. We will show later how this affects Fortran support.

At the same time, it should be noted that the Aurora OS has higher security requirements and blocks dependencies, which can be potentially used for malicious purposes. In particular, this means that custom builds of system libraries cannot be included in the application package [9].

B. Multiple language support

The requirement is that at least English and Russian must be supported. In practice, all speech-to-text (STT) engines are focused on English support. The Russian language is conditioned by the main customers of solutions based on Aurora OS, these are corporations and state companies in the Russian Federation.

Providing and maintaining the model of the local language is a separate relatively huge task [10]–[14]. So the pragmatic approach is to try first the solutions where multi-language

support is already available. The option to form the custom model, vocabulary, or to adapt to the final subject area is also very valuable due to the non-universality of existing solutions and the possible limitations of the battery and processor resources of the mobile equipment used.

C. On-device recognition

The requirement is that it must be possible to use the speech recognition features without a stable network connection. This is due, for example, to the fact that field workers are often in regions without cellular coverage and other opportunities to connect to the Internet.

Besides, some customers have higher requirements for the security of data processing. Therefore, solutions that use external servers for speech processing cannot be applied.

D. Open-source commercial-friendly solution

This requirement actually means two conditions. First, the project must be active. That is, maintainers must contribute all critical comments and bug fixes to its source code fairly promptly. Second, the license under which the project's source code and required resources are provided must allow free use in third-party solutions. If at least one of these conditions is not met, there is no way to build a stable solution using this project.

III. Review of the existing public ASR toolkits

There are several POSIX-compliant ASR systems, all of them have been tested to meet the formulated requirements. The existing reviews and public discussions of such systems were also taken into account [15]. Kaldi and Mozilla's Open Voice projects are mentioned as perspective in the work to find possible alternatives for voice recognition in the browser [16]. According to [17], Kaldi has leading algorithms and data structures and shows high recognition speed and accuracy. According to [18], Kaldi outperforms all the other recognition toolkits (HDecode, Julius, pocketsphinx, Sphinx-4, and Kaldi), providing training and decoding pipelines including the most advanced techniques out of the box.

Many services applicable to operating systems from the consumer market do not meet the requirements of working in a private customer environment. Therefore, the following solutions are excluded from consideration: Google Cloud Speech-to-Text, Apple Speech framework, Wit.ai, Microsoft Bing Voice Recognition, Houndify API, IBM Speech to Text, Yandex SpeechKit, Amazon, Facebook. Nevertheless, it is worth mentioning that there is an Aurora OS compatible solution that uses Yandex Speech Kit to recognize voice commands and control the mobile device [19].

There are also client-server solutions, that could be deployed to a customer private environment. For example, Speech Technology Center (STC, or SpeechPro) in Russia [20]. Such solutions could fit some customer security requirements and reduce the client mobile device requirements due to the need for a solid connection to the network. That could be also a limitation in terms of the possible coverage issues in some exceptional cases.

There is also Jarvisen Translator from IFLYTEK. It is a Chinese company that continuously excels in competitions and provides on-device offline proprietary speech recognition for many languages. Similar lightweight and accurate project Cheetah from Picovoice is also proprietary [15], [21], [22].

Nvidia NeMO project is a toolkit for creating Conversational AI applications. The toolkit comes with extendable collections of pre-built modules and ready-to-use models for [23]:

- Automatic Speech Recognition (ASR);
- Natural Language Processing (NLP);
- Speech synthesis, or Text-To-Speech (TTS).

It is a fresh research project, should be optimized for the modern NVidia GPUs, but not the mobile devices. Existing pre-trained models for the Russian language are not found, only the discussion [24].

The diagram (Fig. 2) shows a general client-server-based solution in the trusted network where possible.

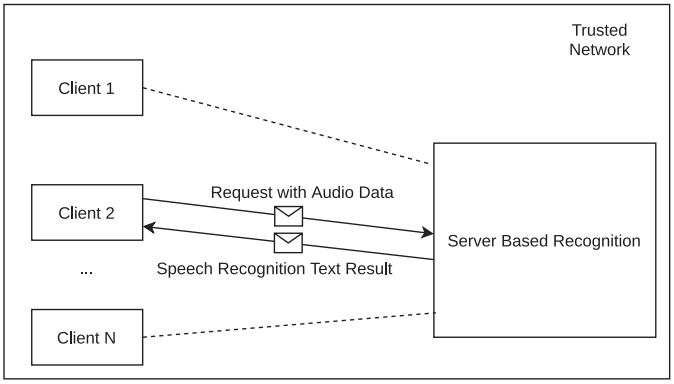

Fig. 2. A general client-server based solution in the trusted network

The best-case option is to be able to recognize the voice on the actual device and open-source solution with public models.

A. CMUSphinx

CMUSphinx project licensed with BSD license, have Russian acoustic model and dictionary. To not be confused with the parts of the toolkit, the documentation provides a list [25]:

1) Pocketsphinx is the lightweight recognizer library implemented using C;
2) Sphinxbase is the support library required by Pocketsphinx;
3) Sphinx4 is the adjustable, modifiable recognizer implemented using Java;
4) Sphinxtrain is the acoustic model training tools.

The main concepts of the project:

- Acoustic model is responsible for matching the sound of the spoken phoneme.
- A dictionary is a file in which lexemes and phonemes are mapped (a word and its transcription). It is needed to

convert phonemes recognized by the acoustic model into lexemes.
- Grammar are formal rules that describe simple rules for constructing sentences. The lexemes obtained in the previous step try to match the grammar and if successful, the result is displayed.
- A language model is a statistical model of a language. It describes the probabilities of words and their combinations. Thus, token recognition is about maximizing the likelihood of a recognized phrase.

The project was suspended in favor of Kaldi project in end of the 2019 [26].

B. Kaldi

Kaldi is a toolkit for speech recognition implemented using C++. Kaldi is similar in aims and scope to Hidden Markov Model Toolkit (HTK is a proprietary software toolkit for handling Hidden Markov Models). The Kaldi goal is to have modern and flexible code, implemented using C++, that is easy to modify and extend. Important features include:

- integration at code level with Finite State Transducers (FSTs) using OpenFst library;
- extensive linear algebra support (via BLAS and LAPACK routines);
- extensible design (the decoder could work from any suitable source of scores, such as a neural net);
- open license (Apache 2.0, which is one of the least restrictive licenses available);
- complete recipes.

The goal of the project is to make available complete recipes for building speech recognition systems, that work from widely available databases such as those provided by the Linguistic Data Consortium (LDC). The projects support linear transforms, MMI, boosted MMI and MCE discriminative training, and also feature-space discriminative training (like fMPE, but based on boosted MMI) [27].

Kaldi has working recipes for Wall Street Journal and Resource Management, and also for Switchboard. The Switchboard recipe is not yet giving state-of-the-art results, due to vocabulary and language model issues, the developers don't use any external data sources for this [27].

Also, the project has a higher-level wrapper. So-called VOSK API and Russian language dataset from the Alpha Cephei company contributors [28]. VOSK API offline speech recognition is based on Kaldi, due to low resource requirements can be used on mobile [15]:

- Supports 17 languages and dialects: English, Indian English, German, French, Spanish, Portuguese, Chinese, Russian, Turkish, Vietnamese, Italian, Dutch, Catalan, Arabic, Greek, Farsi, Filipino.
- Works offline, even on lightweight devices: ARM developer boards, Android and iOS devices.
- Provides Python wrapper.
- Portable per-language models are only 50Mb each, but there are much bigger server models available.
- Provides streaming API for the best user experience.
- There are bindings for different programming languages (Java, C#, JavaScript, etc.).
- Allows quick reconfiguration of vocabulary for best accuracy.
- Supports speaker identification beside simple speech recognition.

The contributions to the OpenFST project from Alpha Cephei developer could also be found: for example, binary search optimizations to improve reachability speed [29].

C. DeepSpeech

DeepSpeech project licensed with Mozilla Public License 2.0. It doesn't have a public Russian language model for now, since it requires a lot of training data to set up the model [30]. Project DeepSpeech uses Google's TensorFlow project to make the implementation easier. It uses a model trained by machine learning techniques based on Baidu's Deep Speech research paper [31]. At the moment of the research step, the Common Voice Mozilla dataset for the Russian language contained about 72 hours of speech. At the moment of the writing, it already contains about 130 hours of the speech. The English Corpus contains about 2000 hours and still growing. There is also Mozilla discourse discussion and experiments from George Fedoseev's dataset for training using YouTube videos with captions, that are not upstreamed: 21% WER on voxforge-ru-test set of Russian speech [32].

Similar techniques are used with Google Android's on-device recognition, while the models are not explicitly public now, there is public research project for learning purposes. It assumes GBoard uses TensorFlow Lite library so the model could be used with TensorFlow and maybe even LWTNN based projects [33].

DeepSpeech also support TensorFlow Lite [34]. The TensorFlow lite is known to run on SailfishOS, the predecessor of the Aurora OS platform. But currently, the Russian language dataset is not enough hours to implement the appropriate recognition as described earlier.

D. Facebook Flashlight

Flashlights ASR application (formerly the wav2letter) provides training and inference capabilities for end-to-end speech recognition systems. Outside of original research conducted with Flashlight and wav2letter, the codebase contains up-to-date implementations of recent architectures and developments in the speech domain.

It is a fresh innovative research project, existing pre-trained models for the Russian language are not found. It is expected to require a lot of computation resources [15].

E. ESPnet

ESPnet is an end-to-end speech processing toolkit, mainly focuses on end-to-end speech recognition and end-to-end text-to-speech. ESPnet uses chainer and pytorch as a main deep learning engine, and also follows Kaldi style data processing, feature extraction/format, and recipes to provide a complete

setup for speech recognition and other speech processing experiments [35]. It is a fresh research project, existing pre-trained models for the Russian language are not found.

IV. The building specificities

Kaldi is the only project that satisfies the requirements of this paper. Therefore, further steps will be described in its context. However, many features, such as the dependence on Fortran, are applicable to other solutions as well.

The main specialty of the Kaldi project is that the developers are not actively maintaining the releases, tags, or versions. The maintainers and lead contributors are also the researchers. They do not want to lose the dynamics of the project supporting any historical versions. So the last version of the repository is used as the current and the only one. And the owners of the repository are not ready to maintain or to back-port any contributions for previous versions to not increase maintenance workload [36]. It could be interpreted as a rolling-release project with the very continuous style of development.

The next thing that needs to be taken into account is the build system. While CMake build system was introduced more than a year ago, handling the third-party libraries for the target Linux distribution could be tricky due to the usage of the internal download tools in the build scripts [37]. For example, the default script of the repository could contain the logic to manually download the exact upstream commit of some of the build dependencies. The maintainers of the distributive-specific packages should take it into account since the build environment options to download any additional files during the build process are limited and should be pre-defined during the start of the process due to security and maintenance reasons.

Also, the future contribution for the Kaldi project could be to introduce pkg-config files to simplify the packaging and linking. Anyway, it's already possible to build the dependencies as shared libraries and manually track the changes and contribute in case of any of the handy build tools are missing in the dependencies.

A. External matrix libraries

There are also dependencies for linear algebra operations. This is a fairly common requirement for all methods and libraries that use machine learning. The matrix code in Kaldi is mostly a wrapper on top of the linear-algebra libraries BLAS and LAPACK [38]:

1) BLAS is a set of subroutine declarations that correspond to low-level matrix-vector operations.
2) CBLAS is the C language interface to BLAS.
3) LAPACK is a set of linear-algebra routines, originally implemented using Fortran. It includes higher-level routines than BLAS, such as matrix inversion, SVD, etc.
4) CLAPACK is a version of LAPACK that has been converted from Fortan to C automatically using the f2c utility.

All the math sobroutins could be implemented via:

- The GNU Scientific Library (GSL), software library in C for numerical computations in applied mathematics and science (not integrated with Kaldi).
- Eigen is the C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms (not replaces LAPACK for Kaldi).
- Intel Math Kernel Library (MKL) is a library of optimized math routines for science, engineering, and financial applications. Its core provides math functions including BLAS, LAPACK, ScaLAPACK, sparse solvers, fast Fourier transforms, and vector math (is not available for ARM).
- Automatically Tuned Linear Algebra Software (Atlas) is the library for linear algebra. It provides an open source implementation of BLAS APIs for C and Fortran77 (not replaces LAPACK for Kaldi).
- OpenBLAS — an open-source (BSD-3-Clause License) implementation of the BLAS and LAPACK APIs with many hand-crafted optimizations for specific processor types. It is developed at the Lab of Parallel Software and Computational Science, ISCAS.
- blasfeo — Basic linear algebra subroutines for embedded optimization (work-in-progress implementation).
- blis — BLAS-like Library Instantiation Software Framework. The framework was designed to isolate essential kernels of computation that, when optimized, immediately enable optimized implementations of most of its commonly used and computationally intensive operations (work-in-progress implementation).

However in practice, some of them are closed-source (not mentioned), some of them are CPU architecture-specific, and some of them are not integrated with Kaldi or not ready yet. So, OpenBLAS is currently the main solution for ARM-based devices with Aurora OS.

B. Fortran support

The main specialty of the modern ASR projects is that linear algebra dependency libraries are dependent on Fortran. And also for the other math-related libraries, like python-numpy and python-scipy. They provide an interface to BLAS and LAPACK for which the reference implementation is done using Fortran [39], [40]. Aurora OS is currently missing Fortran compiler due to build system optimizations. This speeds up the GCC build by a factor of two on armv7l architecture [7]. Such optimization allows to reduce the load on the infrastructure noticeably, considering that the GCC build is a dependency for many other tasks.

However, the exclusion of Fortran from the OS environment and the build tools means that third-party developers cannot use it directly. As stated in the requirements, a toolchain change cannot be done for the already released OS versions. Therefore, the native Fortran support cannot be expected.

At the same time, third-party developers cannot use their own build of the Fortran-enabled gcc compiler to build the entire project, because this will lead to changes in the dependencies of system libraries. And the application package

including the necessary dependencies will be blocked by the security mechanisms.

The project to the automatic port of the LAPACK Fortran code to C code was recently mirrored by VOSK project as a github repository [41]. An attempt to use this method is described in the section VI.

There is also a research project called llvm-fortran, but LLVM is not provided for the Aurora OS repos out of the box.

C. Compilation

To build the Kaldi project in a custom way the appropriate compiler is needed. For example, linaro gcc/gfortran 8.3, same version as the target system (for example, Aurora 3.3.0 target). The QEMU or Aurora SDK VirtualBox image and Docker tools could be used on the x86 machines.

1) Firstly the rootfs should be deployed.
2) OpenBLAS/LAPACK are builded with the make tool specifying the sysroot and the path to the compilers.
3) OpenFST requires autoconf in rootfs to be installed from the repositories (for example, via QEMU). The issue with the Kaldi tools scripts is that build script could not find libfst-ngram library. So the manual build is similar to OpenBLAS with configure and make tools, skipping the tests in the manual build environment similar to Kaldi for Android compilation approach [42]. The mentioned build tools issues could be fixed in the long-term.
4) Finally the Kaldi itself could be built via CMake specifying OpenBLAS as MATHLIB parameter.
5) Simple C language-based example from VOSK API repos could be built after that. With recent updates of the repos, f2c tool could be used to skip the Fortran dependency and convert Fortran-based functions to the C functions. An attempt to use this method is described in the section VI.
6) After that the VOSK API could be used in the end-user Silica-based native Aurora GUI application.

The diagram (Fig. 3) shows the notable runtime and build dependenices of the final client GUI application.

The f2c project ports the LAPACK logic to the C programming language [41]. It is used to solve the issues with the lack of Fortran compiler project. After building all the necessary dependencies, the next libraries is going to be linked with the example project:

`lpthread`, `lfstngram`, `lkaldi-base`, `lkaldi-chain`, `lkaldi-decoder`, `lkaldi-feat`, `lkaldi-cudamatrix`, `lkaldi-fstext`, `lkaldi-gmm`, `lkaldi-hmm`, `lkaldi-ivector`, `lkaldi-kws`, `lkaldi-lat`, `lkaldi-lm`, `lkaldi-matrix`, `lkaldi-nnet`, `lkaldi-nnet2`, `lkaldi-nnet3`, `lkaldi-online`, `lkaldi-online2`, `lkaldi-rnnlm`, `lkaldi-sgmm2`, `lkaldi-transform`, `lkaldi-tree`, `lkaldi-util`.

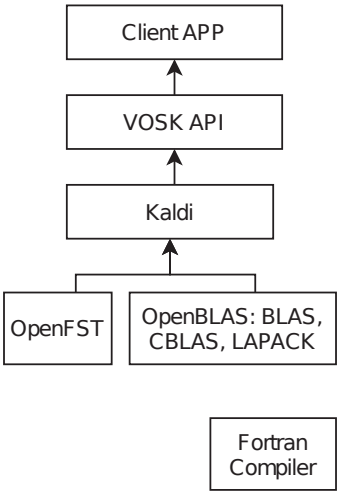

Fig. 3. Notable runtime and build dependencies of the final client application

And the higher-level API of the VOSK project provide three general classes to work with the prepared models: `KaldiRecognizer`, `SpkModel` (speaker model), `Model`. With that classes it is possible to specify the grammar, the direct path to the model files and to work both with the files and the direct speech from the audio device to get both partial and final results of the recognition.

Fig. 4 shows VOSK API classes that could be used in a client application.

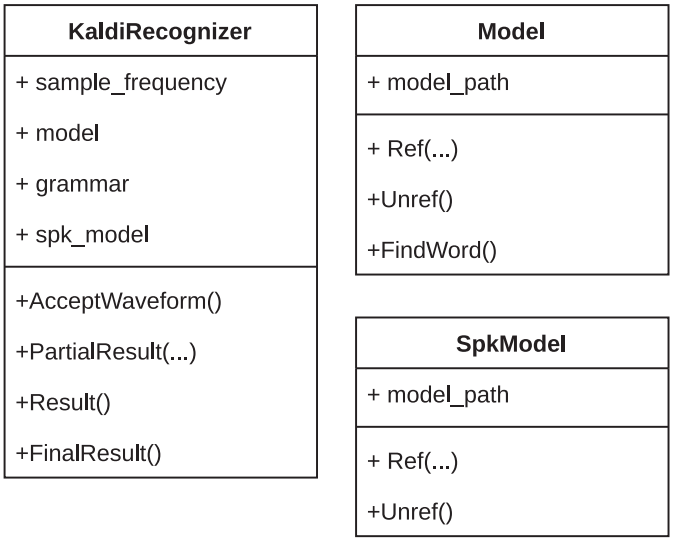

Fig. 4. VOSK API

V. An applying example of the building results

The expected components of the demo project after the building of the libraries are: appropriate test wav file, language model [43], vosk wrapper, kaldi, openfst, openblas/lapack.

The minimal initialization of the VOSK recognizer contains two main steps:

1) Set the source of the Model (path of the model directory).
2) Set sampleFrequency and the model for the `KaldiRecognizer`.

After that `AcceptWaveform()` method of the recognizer could be used to request the data and `Result()`, `PartialResult()` and `FinalResult()` return the results.

There is also a shell script version of the WAV file recognition logic, since the binaries needed for the recognition are also available in the repository. The script could be found for the bigger (server) version of the VOSK models project [43]. The script is based on the binaries of the Kaldi package to decode wav file. The binaries usage is described in the Kaldi documentation. [44]

VI. Discussion

A. Support of the f2c solution

While the f2c tool is simple to use, the issue could be faced at the step where Fortran standard F95 is used in LAPACK instead of F77 (depending on the LAPACK version). f2c could handle only the F77 standard. That is why the VOSK project public mirror of the CLAPACK as the github repository is helpful to discuss the possible issues [41].

During the research, the CLAPACK from the VOSK mirror successfully compiled in Aurora SDK environment, while all the CMake tests from the TESTING directory failed. So, the CLAPACK support for Aurora OS could be proposed as future work and currently could not be compared directly with the gcc-gfortran build from the Linaro toolchain.

In practice, f2c-based solution could show better compiler side optimizations according to f2c public discussion [45]. So the Linaro gfortran compiler (same gcc version as from Aurora SDK) was used to compile the required Fortran code as a shared library to check the solution, while it could not be used in the final applications due to the limitations discussed in the section IV-B.

B. Performance

The test solution is a demo application and INOI R7 device (ARMv7 Qualcomm Snapdragon 212 quad-core processor, chipset MSM-8909 v2 with 2GB RAM). During recognition it was measured that the task performs in less that 10 seconds and require about 100% of first CPU kernel and also about 20% of the second one.

No work was done to optimize memory consumption and startup acceleration by the operating system. Such a study can be qualitatively performed only when the engine is integrated with the OS.

C. Client-server VOSK solution

It is also worth mentioning that the VOSK project can be used to implement a client-server solution. There are four different servers that support four major communication protocols - MQTT, GRPC, WebRTC and Websocket [46].

For example, to communticate via MQTT protocol, it is possible to use any MQTT broker (e.g Mosquitto), prepare and start with an existing VOSK project [47]:

```
#!/bin/bash

# Start ASR server
asr_server_mqtt.py
# Run a test script: ./test_mqtt.py
test_mqtt.py
```

It is also possible to switch from Russian to any other model without restarting the server via sending the MQTT command.

The server-based solution can be used in a trusted network as described in the ASR toolkits review to provide speech recognition to smart deployments or private telephone networks used within a company or organization. The server can also be run as a backend for streaming speech recognition on the cloud, it can power chatbots, web services and telephony.

VII. Conclusion

This article discusses the problem of developing third-party software for Aurora OS in the case when automatic speech recognition function is needed. A set of requirements was formulated for suitable ASR projects, which allow their use on the Aurora OS based devices with commercial projects in Russia. The specified requirements are met only by the Kaldi engine.

Fortran is required to build and use Kaldi. Development tools for current versions of Aurora OS do not support building using Fortran. Therefore, an approach was proposed to build the software with a third-party toolchain and integrate it with the environment. It allows third-party solutions to implement speech recognition on Aurora OS devices.

The proposed approach can be used in other similar cases when the operating system or the build tools do not support Fortran, but this programming language is required to build or run the software. Such cases arise, for example, when math libraries such as LAPACK or OpenBLAS are used. Popular projects that depend on these libraries are PyTorch and TensorFlow.

Acknowledgment

The authors would like to express special thanks to the OpenBLAS and Kaldi ASR Toolkit maintainers and contributors and VOSK API developers: Nikolay V. Shmyrev and Alpha Cephei Inc.

References

[1] Siri overview - apple developer. [Online]. Available: https://developer.apple.com/siri/
[2] Google assistant — google developers. [Online]. Available: https://developers.google.com/assistant
[3] Gboard — google play. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin
[4] Apple developer documentation: On device recognition. [Online]. Available: https://developer.apple.com/documentation/speech/sfspeechrecognitionrequest/3152603-requiresondevicerecognition
[5] Android offline speech recognition natively on pc. [Online]. Available: https://hackaday.io/project/164399-android-offline-speech-recognition-natively-on-pc

[6] Aurora OS website. [Online]. Available: https://auroraos.ru/

[7] Drop Fortran support. on armv7l, this drops gcc compile time from 34minutes to 17minutes (fc433008) · commits · mer-core / gcc · gitlab. [Online]. Available: https://git.sailfishos.org/mer-core/gcc/commit/fc433008bf73e6198739142c5abcf4b67e6a47db

[8] Audio overview — Qt multimedia 5.6. [Online]. Available: https://doc.qt.io/archives/qt-5.6/audiooverview.html

[9] Aurora OS public api. [Online]. Available: https://community.omprussia.ru/documentation/software_development/reference/public_api.html

[10] N. Gabriel, "Automatic speech recognition in somali," 2020.

[11] A. Ali, Y. Zhang, P. Cardinal, N. Dahak, S. Vogel, and J. Glass, "A complete kaldi recipe for building arabic speech recognition systems," in *2014 IEEE spoken language technology workshop (SLT)*. IEEE, 2014, pp. 525–529.

[12] Y. G. Thimmaraja and H. Jayanna, "Creating language and acoustic models using Kaldi to build an automatic speech recognition system for kannada language," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE, 2017, pp. 161–165.

[13] J. Guglani and A. N. Mishra, "Continuous punjabi speech recognition model based on Kaldi ASR toolkit," *International Journal of Speech Technology*, vol. 21, no. 2, pp. 211–216, 2018.

[14] A.-L. Georgescu, H. Cucu, and C. Burileanu, "Kaldi-based DNN architectures for speech recognition in romanian," in *2019 International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*. IEEE, 2019, pp. 1–6.

[15] N. V. Shmyrev. (2020) What are the top ten speech recognition apis? [Online]. Available: https://www.quora.com/What-are-the-top-ten-speech-recognition-APIs

[16] D. Gonze and O. Bonaventure, "Coding with the voice," *École polytechnique de Louvain: Master in Civil Engineering*, 2020.

[17] M. Belenko and P. Balakshin, "Comparative analysis of speech recognition systems with open code," *Mezhdunarodnyy nauchno-issledovatel'skiy zhurnal*, no. 04 (58) Part 4, pp. 13–18, 2017.

[18] C. Gaida, P. Lange, R. Petrick, P. Proba, A. Malatawy, and D. Suendermann-Oeft, "Comparing open-source speech recognition toolkits," in *11th International Workshop on Natural Language Processing and Cognitive Science*, 2014.

[19] P. Vytovtov, "Voice assistant for sailfish os," *Proceeding of the 20th conference of FRUCT Association*, pp. 741–742, 2017.

[20] Speech technology center. [Online]. Available: https://speechpro.com/

[21] Jarvisen translator. [Online]. Available: https://www.iflytek.com/en/products/#/translat

[22] Picovoice: Edge voice ai platform. [Online]. Available: https://picovoice.ai/

[23] O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook *et al.*, "Nemo: a toolkit for building AI applications using neural modules," *arXiv preprint arXiv:1909.09577*, 2019.

[24] Nvidia NeMo: Training non-english asr model. [Online]. Available: https://github.com/NVIDIA/NeMo/issues/234

[25] Overview of the CMUSphinx toolkit – CMUSphinx open source speech recognition. [Online]. Available: https://cmusphinx.github.io/wiki/tutorialoverview/

[26] Update on CMUSphinx project. [Online]. Available: https://cmusphinx.github.io/2019/10/update/

[31] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[27] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The kaldi speech recognition toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011, iEEE Catalog No.: CFP11SRW-USB.

[28] Vosk offiline speech recognition api, Kaldi based. [Online]. Available: https://alphacephei.com/vosk/

[29] OpenFST: Reachability speed optimization. [Online]. Available: https://github.com/alphacep/openfst/commit/65e9de91b35a7238ffa035ed8bb2594a50dbdc5a

[30] Mozilla DeepSpeech: Adapting engine to any custom language issue. [Online]. Available: https://github.com/mozilla/DeepSpeech/issues/692

[32] George fedoseev: DeepSpeech with YouTube captions dataset. [Online]. Available: https://github.com/GeorgeFedoseev/DeepSpeech

[33] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, "A comparison of sequence-to-sequence models for speech recognition," in *Interspeech*, 2017, pp. 939–943.

[34] Deepspeech 0.6: Mozilla's Speech-to-Text engine gets fast, lean, and ubiquitous - mozilla hacks - the web developer blog. [Online]. Available: https://hacks.mozilla.org/2019/12/deepspeech-0-6-mozillas-speech-to-text-engine/

[35] H. Inaguma, S. Kiyono, K. Duh, S. Karita, N. Yalta, T. Hayashi, and S. Watanabe, "ESPnet-ST: All-in-one speech translation toolkit," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Online: Association for Computational Linguistics, Jul. 2020, pp. 302–311. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-demos.34

[36] Kaldi: Change log / release issue. [Online]. Available: https://github.com/kaldi-asr/kaldi/issues/1943

[37] Kaldi: Build system issue. [Online]. Available: https://github.com/kaldi-asr/kaldi/issues/3086

[38] Kaldi: External matrix libraries. [Online]. Available: https://kaldi-asr.org/doc/matrixwrap.html

[39] Quora: Which python packages besides Numpy use Fortran? [Online]. Available: https://www.quora.com/Which-python-packages-besides-Numpy-use-fortran

[40] Researchgate: Why are physicists stuck with Fortran? [Online]. Available: https://www.researchgate.net/post/Why-are-physicists-stuck-with-Fortran-and-not-willing-to-move-to-Python-with-NumPy-and-Scipy

[41] Vosk project: CLAPACK. [Online]. Available: https://github.com/alphacep/clapack

[42] E. Silva. (2017) Compile Kaldi for Android. [Online]. Available: https://jcsilva.github.io/2017/03/18/compile-kaldi-android/

[43] VOSK models. [Online]. Available: https://alphacephei.com/vosk/models

[44] Kaldi documentation. [Online]. Available: https://kaldi-asr.org/doc/online2-wav-nnet3-latgen-faster_8cc.html

[45] c - how much better are fortran compilers really? - computational science stack exchange. [Online]. Available: https://scicomp.stackexchange.com/questions/1194/how-much-better-are-fortran-compilers-really

[46] Websocket, gRPC, MQTT and WebRTC servers for VOSK. [Online]. Available: https://alphacephei.com/vosk/server

[47] Vosk ASR via MQTT. [Online]. Available: https://github.com/alphacep/vosk-server/tree/master/mqtt