Hera - An Operating System Level Voice Recognition Package

CS492 Project

Project Report
submitted by

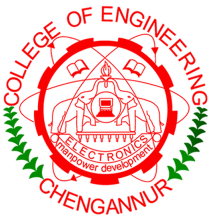
CHN18CS027 Arjun Vishnu Varma
CHN18CS045 Gokul Manohar
CHN18CS066 Jithin James
CHN18CS086 Nandu Chandran

to

APJ Abdul Kalam Technological University
in partial fulfilment of the requirements for the award of the Degree

Bachelor of Technology

Computer Science & Engineering



Department of Computer Engineering
College of Engineering Chengannur
Alappuzha 689121
Phone: +91.479.2451424
ceconline.edu
hod.cse@ceconline.edu

May 2022

Declaration

We undersigned hereby declare that this project report titled Hera - An Operating System Level Voice Recognition Package, submitted for partial fulfillment of the requirements for the award of the degree Bachelor of Technology of APJ Abdul Kalam Technological University, Kerala is a bonafide work done by us, under the supervision of Associate Professor Ahammed Siraj K K. This submission represents our ideas in our own words and where ideas or words of others have been included, we have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other university.

Place: Chengannur

Date: June 28, 2022

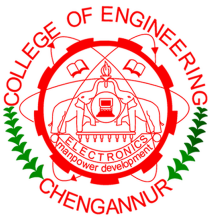
Arjun Vishnu Varma

Gokul Manohar

Jithin James

Nandu Chandran

College of Engineering Chengannur
Department of Computer Engineering



Certificate

This is to certify that, this project report titled **Hera - An Operating System Level Voice Recognition Package** is a bona fide record of the CS492 Project done by Eight Semester B.Tech. Computer Science & Engineering students **Arjun Vishnu Varma, Gokul Manohar, Jithin James & Nandu Chandran** under our guidance and supervision, in partial fulfilment of the requirements for the award of the degree, Bachelor of Technology in Computer Science and Engineering of **APJ Abdul Kalam Technological University.**

Guide

Ahammed Siraj K K

Associate Professor

Computer Engineering

Coordinator

Shiny B

Assistant Professor

Computer Engineering

Head of the Department

Dr. Manju S Nair

Associate Professor

Computer Engineering

Acknowledgements

We are greatly indebted to God Almighty for being the guiding light throughout with his abundant grace and blessings that strengthened us to do this endeavour with confidence. I express our heartfelt gratitude towards our guide Associate Professor Ahammed Siraj K K. Also I express our sincere gratitude towards Dr. Smitha Dharan, Principal, College of Engineering Chengannur for extending all the facilities required for doing our project. We would also like to thank Dr. Manju S Nair, Head of the Department of Computer Engineering, for providing constant support, encouragement and guiding us in doing this project. Now we extend our sincere thanks to our project coordinators Ms. Shiny B, Assistant Professor in Computer Engineering, and Ms. Sreelekshmi K R, Assistant Professor in Computer Engineering for guiding us in our work and providing timely advice and valuable suggestions. Last, but not the least, we extend our heartfelt gratitude to our parents and friends for their support and assistance.

Place: Chengannur

Date: June 28, 2022

Arjun Vishnu Varma

Gokul Manohar

Jithin James

Nandu Chandran

Abstract

This project presents Hera, an Operating System level voice recognition package that understands voice commands to perform actions to simplify the user's workflow. We propose a modernistic way of interacting with Linux systems, where the latency of conventional physical inputs are minimized through the use of speech recognition. Currently, voice assistant rely heavily on a primary server to perform most of the computation. Though this results in faster processing, this is still a privacy concern because we are not explicitly known how our data is used. Contrary to this, data processing in Hera is done locally, on the client side. The voice data that is gathered, is sent to a speech recognition engine, and a natural language processing model is able to draw meaningful conclusions from it to perform the service requested by the user.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the advancements in Computer Science, the human-computer interaction have seen a dramatic rise in voice based inputs [1]. The use of natural language to communicate with the system means faster inputs, and improved user experience. Speech has the potential to provide a direct and effective alternative to the traditional keyboard and mouse based input in a desktop environment. Added, the visually challenged people can also benefit from the voice based system interactions. So there are many benefits in developing a human-computer interface that utilizes voice interactions. We can incorporate voice interactions to a system using either online or offline approach, where online based is the most common one. But the problem with the online based approach is that, you have to be connected to the internet for it to work and also, it relies heavily on a primary server to perform most of the computation on the captured speech data to draw its meaning. This is a privacy concern. Because, we are not explicitly known how our speech data is handled, processed or stored. So in order to respect privacy, the best option available is the offline based speech recognition. In this approach, the speech data is captured through the microphone, and it is then processed on the user side itself.

Though there are voice assistants or Virtual Assistants (VA) available on smartphones and some desktop platforms like Windows and macOS, they are not fully able to interact with the operating system and are not much customizable. They are also online based. So we identified that, there was a need for an offline based operating system level speech recognition system that can be a replacement for the online voice assistants.

1.1 Proposed Project

Our project propose a new way of interacting with the operating system that prioritizes on improving the user experience via voice commands. It is able to recognize the spoken natural language and is able to draw meaningful conclusions from it and to provide responses accordingly. Unlike the traditional approach which rely heavily on the physical inputs, our system can provide an alternative method through the means of voice interactions. Though we are developing a voice based system, the traditional physical input is still available, so the user can experience the best of both worlds. Also, we are developing this software to work offline, so the speech data is processed on the user's system itself, which in turn improves the privacy. In order to have a better integration with the operating system, the software was developed to work on the Linux platform.

1.1.1 Problem Statement

Enhancing the user experience of Linux Operating System by adding voice recognition.

1.1.2 Proposed Solution

For effective and efficient embedding of speech recognition into Linux Operating System, we employ a multimodule approach, namely 'Voice', 'OS level Coordinator' and 'Skill' modules. These modules determine how the voice data is collected, and evaluated. The entire working of the system is divided into two phases, Voice-Coordinator (Primary) phase and Coordinator-Skill (Secondary) phase. The primary phase consist of transcribing the voice data to the corresponding intents. The secondary phase deals with mapping intents into corresponding skills and providing feedback in the form of speech or text data.

Chapter 2

Report of Preparatory Work

In the research for our project, we found out that an offline Automatic Speech Recognition (ASR) system can be implemented on Linux using the approaches listed below.

2.1 Literature Survey

2.1.1 Pocketsphinx

Pocketsphinx [2] is a part of the CMU Sphinx Open Source Toolkit For Speech Recognition. CMU Sphinx, also called Sphinx for short, is the general term to describe a group of speech recognition systems developed at Carnegie Mellon University. These include a series of speech recognizers (Sphinx 2 - 4) and an acoustic model trainer (SphinxTrain). In 2000, the Sphinx group at Carnegie Mellon committed to open source several speech recognizer components, including Sphinx 2 and later Sphinx 3 (in 2001). The speech decoders come with acoustic models and sample applications. The available resources include in addition software for acoustic model training, language model compilation and a public domain pronunciation dictionary, cmudict.

2.1.2 Kaldi

Kaldi [3] is an open-source toolkit for speech recognition written in C++ and licensed under the Apache Licence v2.0. It provides a speech recognition system based on finite-state transducer (using the freely available OpenFst). Kaldi is capable of generating features like MFCC, fbank, fMLLR, etc. Hence, in recent deep neural network research, a popular usage of Kaldi is to pre-process raw waveform into acoustic feature for end-to-end neural models.

2.1.3 HTK

Hidden Markov Toolkit (HTK) is a group of programs and libraries used to develop ASR with the Hidden Markov model. Actually, HTK itself is complete to build the whole ASR starting from speech database, feature extraction, construction and training of acoustic models and conducting test offline and real-time online. [4]

2.1.4 DeepSpeech

DeepSpeech is an open source Speech-To-Text engine, using a model trained by machine learning techniques based on Baidu's Deep Speech research paper [5]. Project DeepSpeech uses Google's TensorFlow to make the implementation easier.

For our project, we opted for Kaldi based speech recognition system because of it's low Word Error Rate (WER) [6] and the ease of implementation.

Chapter 3

Project Design

3.1 Methodology

We require a system that can identify spoken speech It is necessary to transform the spoken voice to text. The exact 'keyword' or 'skill word' must now be found. The user's requested specified action is then carried out by us. Additionally, the machine must awaken to a particular 'Wake Word'.

The system does not have to be in constant listening mode. Speech recognition could be used for this, but it is computationally intensive, adds a lot of latency, and drains power, especially if the machine is portable and powered by batteries. We will employ a 'Wake Word' neural network to awaken the system in order to address this issue. We implemented a 4-step process in developing Hera.

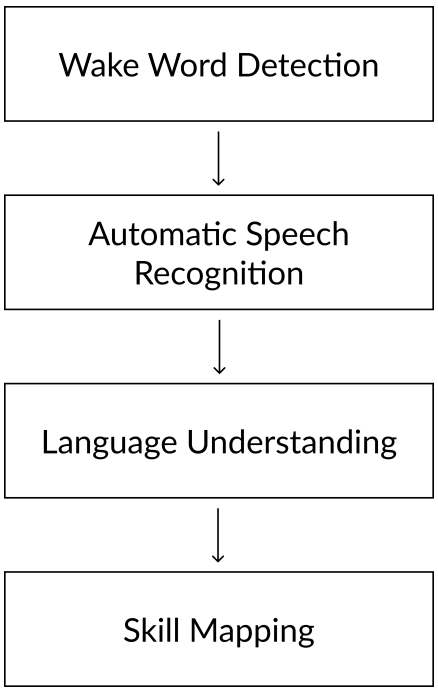


Figure 3.1: Methodology

3.1.1 Wake Word Detection

Wake word detection is the task of recognizing an utterance for activating the system to process the voice inputs [7], such as "Hey, Alexa" for the Amazon Echo. Given that such systems are meant to support fully automatic speech recognition, the task seems simple. However, it introduces a different set of challenges because these systems have to be always listening, computationally efficient, and, most of all, privacy respecting. Therefore, researchers treat it as a separate line of work, with most recent advancements driven by neural networks. For the application of the detected wake word in ASR, the problem of improving speech recognition with the help of the detected wake word is investigated.

3.1.2 Automatic Speech Recognition

Automatic Speech Recognition or ASR, as it's known in short, is the technology that allows human beings to use their voices to speak with a computer interface in a way that, in its most sophisticated variations, resembles normal human conversation. For achieving this, we use a Kaldi [8] recognizer model. The most advanced version of currently developed ASR technologies revolves around what is called Natural Language Processing, or NLP in short.

3.1.3 Language Understanding

Natural Language Understanding (NLU) or Natural Language Interpretation (NLI) [9] is a subtopic of Natural Language Processing (NLP) in artificial intelligence that deals with machine reading comprehension. For achieving this, we implemented an Intent classification model and an Entity extractor model. Intent classification is the automated categorization of text data based on customer goals. In essence, an intent classifier automatically analyses texts and categorizes them into intents. Entity extraction is a text analysis technique that uses Natural Language Processing (NLP) to automatically pull out specific data from unstructured text, and classifies it according to predefined categories. These categories are named entities, the words or phrases that represent a noun.

3.1.4 Skill Mapping

Skills are a pre-programmed set of actions that Hera will perform once the intent of the user is identified. For example, skills of Hera can be launching applications, file management (to copy and move files between different locations), media playback (play required media files). Hera evolves as more skills are added.

3.2 Hardware Requirements

The minimum hardware configuration required for the proper functioning of the system is outlined below.

- CPU: Intel Core i3 or above.
- Operation Speed: 2.0 GHz or above.
- RAM: 4 GB or above.
- Hard Disk: 50 GB or above.
- Speakers: Speech Synthesis

3.3 Software Requirements

The minimum software configuration required for the proper functioning of the system is outlined below.

- Operating System: Any Linux distribution.
- Development language: Python 3.7.

3.3.1 Languages

Python

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. [10]

Shell and Bash

Certain scripts are written using shell or bash programming languages.

YAML and JSON

Databases and configurations are implemented in YAML and JSON.

3.3.2 Tools

TensorFlow

TensorFlow is a software library, developed by Google Brain Team within Google's Machine Learning Intelligence research organization, for the purposes of conducting machine learning and deep neural network research. Wake word detection is implemented using TensorFlow.

Keras

Keras is a compact, easy to learn, high-level Python library run on top of TensorFlow framework. It is made with focus on understanding deep learning techniques, such as creating layers for neural networks maintaining the concepts of shapes and mathematical details. Keras is used to implement wake word model.

IDE

Sublime Text & Visual Studio Code are cross-platform source code editor that we used for our development. It natively supports many programming languages and markup languages.

3.3.3 Machine Learning Models

Vosk

Vosk is an offline open source speech recognition toolkit. It enables speech recognition for 20+ languages and dialects. Vosk models are small but provide continuous large vocabulary transcription, zero-latency response with streaming API, reconfigurable vocabulary and speaker identification. Vosk supplies speech recognition for chatbots, smart home appliances, virtual assistants. Vosk uses Kaldi recognizer to implement the automatic speech recognizer in its core. The ASR for our project is implemented using Vosk.

SGDClassifier

The intent of the text converted from the speech is recognized by passing it through an Intent Classifier. SGD Classifier [11] is a linear classifier (SVM, logistic regression) optimized by the Stochastic Gradient Descent (SGD).

spaCy - Named Entity Recognition

The entity inside the text converted from the speech is extracted by using an entity extractor model. For this, we make use of spaCy's [12] Custom Named Entity Recognition (NER).

3.3.4 Speech Synthesis

eSpeak

For Linux and Windows, eSpeak is a small open source speech synthesizer for English and other languages. eSpeak uses 'formant synthesis' approach. This makes it possible to supply several languages in a compact footprint. The voice is understandable and fast-moving, but it lacks the naturalness and smoothness of larger synthesizers that are built on recordings of actual speech. eSpeak is available as, a command line program (Linux and Windows) to speak text from a file or from stdin.

Nix-TTS

Nix-TTS[13], a lightweight neural TTS model achieved by applying knowledge distillation to a powerful yet large-sized generative TTS teacher model. Distilling a TTS model might sound unintuitive due to the generative and disjointed nature of TTS architectures, but pre-trained TTS models can be simplified into encoder and decoder structures, where the former encodes text into some latent representation and the latter decodes the latent into speech data. Nix is our primary speech synthesis model.

Kivy - Graphical User Interface Framework

Kivy is an open-source and Graphical User Interface (GUI) development platform for Python. It helps us to develop mobile applications and multitouch application software with a NUI (Natural User Interface) It allows developers to build an application once and use it across all devices. We used Kivy for implementing the GUI for our Hera.

Table 3.1: Tools at a glance

Tools	Context
TensorFlow	Wake Word Detection
Keras	Wake Word Detection
Vosk	Automatic Speech Recognition
SGDClassifier	Language Understanding
spaCy NER	Language Understanding
eSpeak	Speech Synthesis
nix-TTS	Speech Synthesis
Sublime	IDE
VSCode	IDE

3.4 Working Principle

As stated in the proposed solution, we employ a multimodule approach, namely Voice, Coordinator and Skill modules. The working is in two phases.

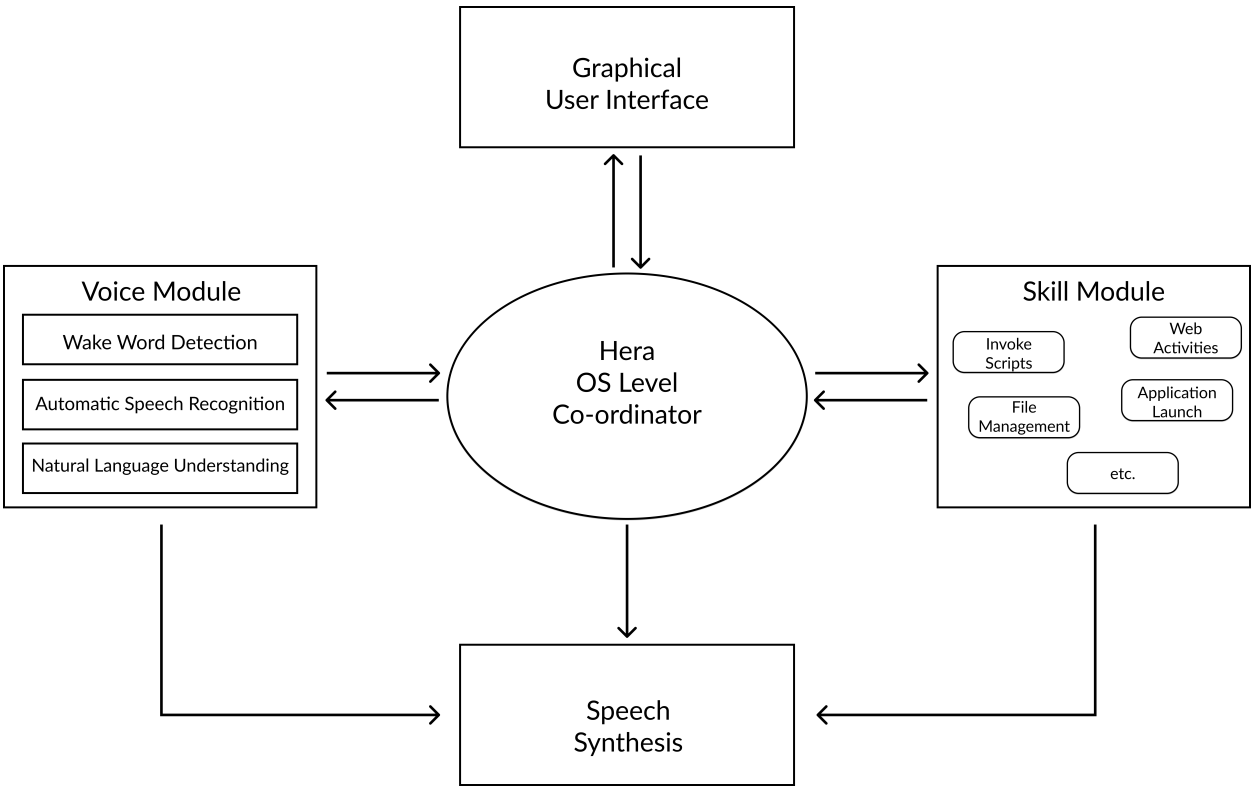


Figure 3.2: Working principle diagram

First is the Voice-Coordinator phase. In this module, wake word gets detected and this triggers automatic speech recognizer to capture the speech and convert it to text. Finally, NLU draws the meaning and intent behind the text.

Next is the Coordinator-Skill phase. Here, the mapped skill gets called. The speech feedback from the Voice module, Coordinator and the Skill module is synthesized by a TTS engine. The whole user interaction takes place in the form of speech as well as Graphical User Interface (GUI) texts.

Chapter 4

Implementation

4.1 Activating Hera using Wake Word

The typical workflow for interacting with a voice assistant is to first activate it with a 'wake' or 'hot' word, then provide your voice command. We have made a wake word detection convolutional neural network using TensorFlow. We take a moving window (a time slice) of the given audio file and compute the Fast Fourier Transform (FFT) of that time slice. That gives us an indication of power at each frequency component for that time slice. We then filter that power graph using a Mel-spaced filter bank.

The energy under each of these filters is added up to create a vector of numbers. The logarithm of these numbers is computed, and the discrete cosine transform (DCT) is then performed on the vector. The output of these operations provides us with the Mel Frequency Cepstral Coefficient (MFCC) terms. We then slide the window over to compute all the MFCCs for that particular audio sample. Convolution is used for filtering the image to identify features in that image. In our case, we use a CNN model for performing image classification. After convolution, we perform the rectified linear unit (ReLU) activation function. We then perform a Max Pooling operation, we're using a pre-made CNN to help classify the MFCC features from spoken words. Then we add an optimizer called 'Adam' to predict the answer. MFCC calculations, model shape and size, number of epochs, etc. are considered hyperparameters. These are values that are set prior to training and are not automatically updated during training.

4.2 Speech to Text using vosk ASR

After the wake word has been detected, Hera is able to receive the speech data from the user. For that, firstly, the vosk model is loaded to reduce the output lag when the recognizer is called. We are using vosk-model-en-in-0.4, a 727 MB model.

We could use a bigger model to improve the accuracy, but the RAM usage will also be higher and could cause lag on prediction. Then the sound device is enabled and is used to capture the speech data. The speech data is sent to the `vosk.KaldiRecognizer()` to convert the speech to text.

The working of Kaldi based vosk can be explained in two steps.

4.2.1 Preprocessing and Feature Extraction

This part deals with pixel-based representation for the speech data. Kaldi uses MFCC (Mel-Frequency Cepstral Coefficients) for this representation. Along with MFCC, Kaldi uses two more features, such as CMVN (Cepstral Mean And Variance Normalization) and I-Vectors.

- MFCC and CMVN are used for representing the content of each audio utterance.
- I-Vectors are used for representing the style of each audio utterance or speaker.

4.2.2 Model

Kaldi's model can be divided into two main components:

Acoustic Model

The first part is the *Acoustic Model*, which is a Deep neural network. That model will transcribe the audio features that we created into some sequence of context-dependent phonemes.

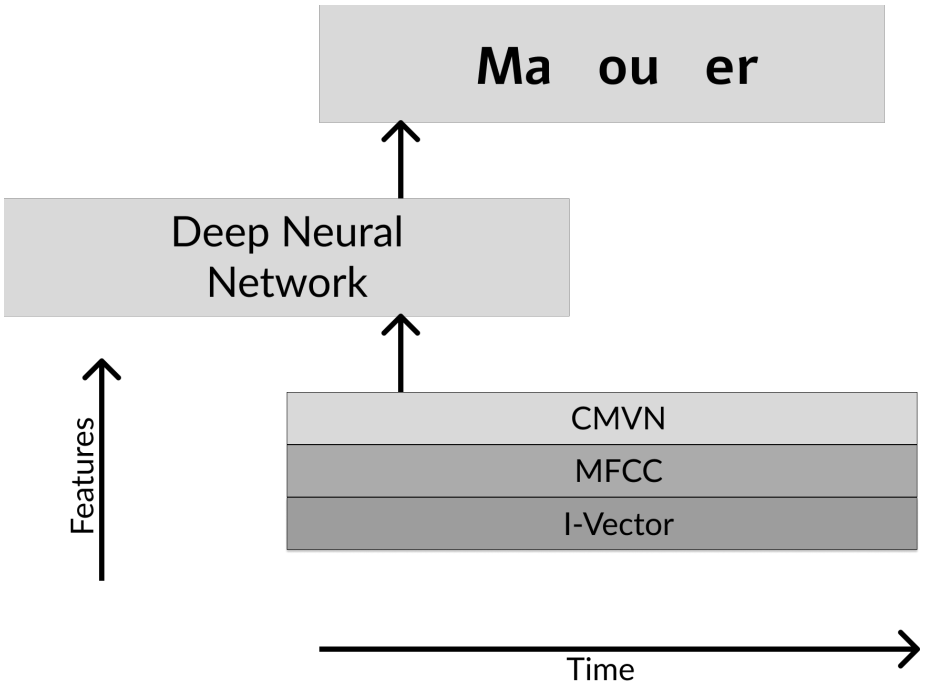


Figure 4.1: Kaldi Acoustic Model

Decoding Graph

The second part is the *Decoding Graph*, which takes the phonemes and turns them into lattices. A lattice is a representation of the alternative word-sequences that are likely for a particular audio part.

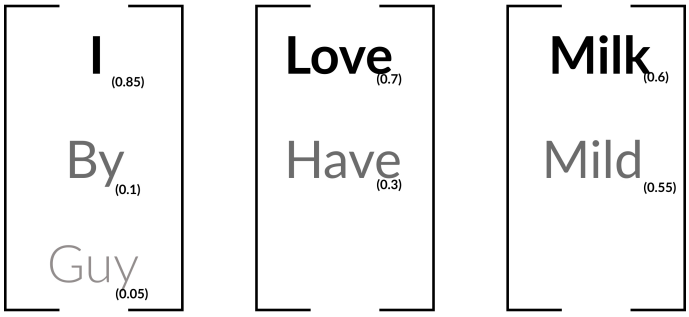


Figure 4.2: A representation of lattice

Finally, the resulting text combination is returned to the Coordinator.

4.3 Intent Classification using SGDClassifier

The text obtained from the speech recognizer is passed to the Intent Classifier. The classifier model used in our project is called SGDClassifier. The pipeline is like this

```
Pipeline(
    [
        ('tfidf', TfidfVectorizer()),
        ('sgd', SGDClassifier())
    ]
)
```

The intents are custom trained using a JSON file with text mapped to intents like this,

```
"play some music": "MUSIC_PLAYBACK_RANDOM_SONG",
"play any music": "MUSIC_PLAYBACK_RANDOM_SONG",
"open the terminal": "LAUNCH_APPLICATION",
"open terminal": "LAUNCH_APPLICATION"
```

The text is vectorized using `TfidfVectorizer`. Then it is classified with respect to trained model. The classification is done using SGDClassifier. This estimator implements regularized linear models with stochastic gradient descent (SGD) learning. The output of the text classification process is an intent derived from the existing model. Finally, the result is returned to the Coordinator.

4.4 Skill Mapping

The intent is used to map to a specific skill. These skills are called, and the text command is passed as the argument. We provide some example skills like

- Music Playback
- Application Launching
- Website Launching
- File Management
- Custom scripts

We could always add more skills in the future to expand Hera's skill set. The outputs from skills are in the form of text (displayed in GUI) and as voice feedbacks.

4.5 Entity Extraction using spaCy NER

The Named Entity Recognition is used at the start of the skill mapping. The text command received as the skill argument is passed to the spaCy NER model. The spaCy does use word embeddings for its NER model, which is a multilayer CNN. Word embeddings are a technique for representing text where different words with similar meaning have a similar real-valued vector representation. spaCy is trained with training and validation data for creating a custom model. When the text received, firstly the custom trained model is loaded `spacy.load(model_path)` and the text is checked for embeddings and a suitable match is obtained. This is returned to the skill module.

4.6 Voice output using Nix-TTS

The output from the skills are provided to the user through the use of voice feedback. For that, we use a module for Text to Speech for Linux called espeak. However, the speech generated is quite robotic and unpleasant. But there is a new approach for improving the speech quality. Nix-TTS is a new project that works by applying knowledge distillation to a powerful yet large-sized generative TTS teacher model. The result of this is an incredibly lightweight, human-like end-to-end Text-to-Speech Model.

4.7 Graphical User Interface using Kivy

For a non-terminal based input/output, we designed a front end application which can display the text output in real-time. This can elevate the user experience as it makes the interactions easier while providing abstraction. Kivy also provide a material design library called KivyMD.

Chapter 5

Results and Conclusions

5.1 Results

We were required to execute two phases, namely Voice-Coordinator and Coordinator-Skill, during the construction of our system. The goal of a Voice-Coordinator phase is to gather voice data, comprehend it, and relay it to the coordinator. The Coordinator-Skill converts the preceding output into the necessary skill. Both features were successfully combined. The following features are ones we've implemented successfully in our short time frame.

1. Wake Word Detection
2. Automatic Speech Recognition
3. Intent Classification
4. Skill Mapping
5. Entity Extraction
6. Voice feedback
7. GUI development

Here are some snapshots of the working.

[Snapshot] Wake Word Detection

The Wake Word was successfully identified by our system.

```
(env) gm@zorinos:~/Hera$ python wake_word_detection/wake_word_detection_script.py
2022-06-27 23:52:01.760639: W tensorflow/stream_executor/platform/default/dso_load
 object file: No such file or directory
2022-06-27 23:52:01.760670: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] I
2022-06-27 23:52:03.323969: E tensorflow/stream_executor/cuda/cuda_driver.cc:271]
2022-06-27 23:52:03.324028: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:
t exist
2022-06-27 23:52:03.324525: I tensorflow/core/platform/cpu_feature_guard.cc:193] T
 instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compil
Press Enter to start:
[WWD] Pass #1
Waiting for wake word
1/1 [==============================] - 0s 101ms/step
Wake word detected

Complete
(env) gm@zorinos:~/Hera$
```

Figure 5.1: [Snapshot] Wake Word Detection

[Snapshot] Automatic Speech Recognition

Here, the spoken command "Open the terminal" is recognized using automatic speech recognition.

```
(env) gm@zorinos:~/Hera$ python automatic_speech_recognition/automatic_speech_recognition_script.py
LOG (VoskAPI:ReadDataFiles():model.cc:213) Decoding params beam=13 max-active=7000 lattice-beam=6
LOG (VoskAPI:ReadDataFiles():model.cc:216) Silence phones 1:2:3:4:5:6:7:8:9:10
LOG (VoskAPI:RemoveOrphanNodes():nnet-nnet.cc:948) Removed 1 orphan nodes.
LOG (VoskAPI:RemoveOrphanComponents():nnet-nnet.cc:847) Removing 2 orphan components.
LOG (VoskAPI:Collapse():nnet-utils.cc:1488) Added 1 components, removed 2
LOG (VoskAPI:ReadDataFiles():model.cc:248) Loading i-vector extractor from automatic_speech_recognitio
LOG (VoskAPI:ComputeDerivedVars():ivector-extractor.cc:183) Computing derived variables for iVector ex
LOG (VoskAPI:ComputeDerivedVars():ivector-extractor.cc:204) Done.
LOG (VoskAPI:ReadDataFiles():model.cc:279) Loading HCLG from automatic_speech_recognition/vosk-models/
LOG (VoskAPI:ReadDataFiles():model.cc:294) Loading words from automatic_speech_recognition/vosk-models
LOG (VoskAPI:ReadDataFiles():model.cc:310) Loading subtract G.fst model from automatic_speech_recognit
LOG (VoskAPI:ReadDataFiles():model.cc:312) Loading CARPA model from automatic_speech_recognition/vosk-
Press Enter to start:
I'M LISTENING
open the terminal
(env) gm@zorinos:~/Hera$
```

Figure 5.2: [Snapshot] Speech Recognition

[Snapshot] Graphical User Interface

This is the base Graphical User Interface which can display real time output from Hera.

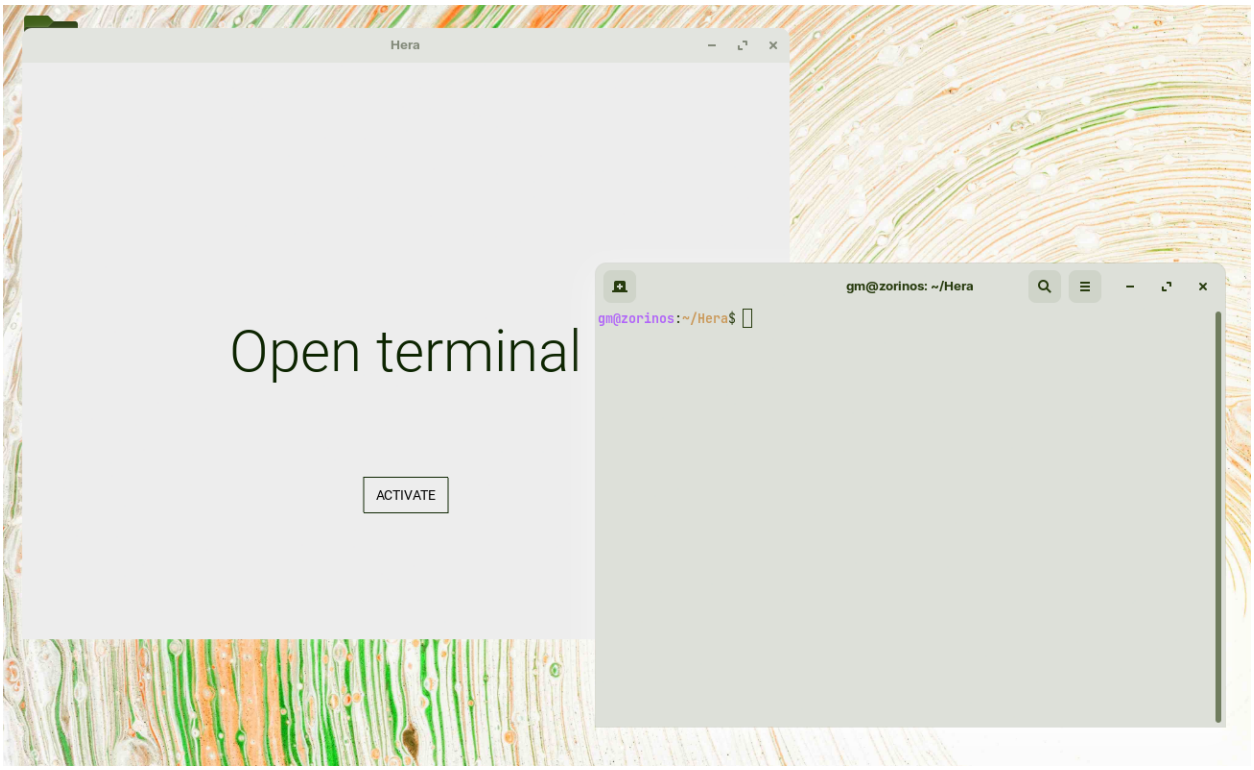


Figure 5.3: [Snapshot] Graphical User Interface

5.1.1 Limitations

These are some limitations we identified in our project.

- The speech recognition software algorithms are not 100% accurate.
- Smaller models mean less accurate speech to text conversion.
- Machine Learning based operations take some time to load and work.
- Microphone is sometimes unable to pick up the speech data.

5.2 Conclusions

Modern voice assistant systems require constant internet connectivity and servers for speech recognition, and the voice data is sent to those systems for processing. Though these systems are faster in processing data, privacy is still a major concern. Keeping this in mind, we have decided to create an OS level voice recognition package for Linux platform called Hera that helps in eliminating all the hassles we discussed above to some extent. In the era of modern assistants, we hope our project Hera stands out by providing a private and hassle-free user experience. Although it is a huge project, we were able to complete a fraction of it in a given time. We showcased the user experience can be improved through the use of voice interactions and that offline speech recognition can be a replacement for the online voice assistants.

5.2.1 Future Work

Hera could be used as an IOT application if we can integrate it with portable devices like the Raspberry Pi. The skills that we demonstrated are just the tip of the iceberg, and the potential is unlimited as we add more skills. So, we are open sourcing this project, so other developers could learn and contribute. The GitHub link of this project is **https://github.com/HeyHera/Hera**.

References

[1] Digital voice assistants in use to triple to 8 billion by 2023, driven by smart home devices. https://www.juniperresearch.com/press/digital-voice-assistants-in-use-to-8-million-2023. Date: 2018 Feb 12.

[2] D. Huggins-Daines, M. Kumar, A. Chan, A.W. Black, M. Ravishankar, and A.I. Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, pages I–I, 2006.

[3] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlíček, Yanmin Qian, Petr Schwarz, Jan Silovský, Georg Stemmer, and Karel Vesel. The kaldi speech recognition toolkit. *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, 01 2011.

[4] Zulkarnaen Hatala. Practical speech recognition with htk, 2019.

[5] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition, 2014.

[6] Automatic speech recognition (asr) systems compared. https://medium.com/sciforce/automatic-speech-recognition-asr-systems-compared-6ad5e54fd65f. Date: 2021 Jul 7.

[7] Yiming Wang. Wake word detection and its applications. *Theses and Dissertations, Electronic (ETDs)*, pages 1–198, 2021.

[8] How to start with kaldi and speech recognition. https://towardsdatascience.com/how-to-start-with-kaldi-and-speech-recognition-a9b7670ffff6. Date: 2018 Nov 22.

[9] Paul Semaan. Natural language generation an overview. *LACSC – Lebanese Association for Computational Sciences*, pages 1–8, 2012.

[10] Dave Kuhlman. *A Python Book*. Dave Kuhlman, 2013.

[11] Sgdclassifier. https://scikit-learn.org/stable/modules/sgd.html.

[12] spacy. https://spacy.io/.

[13] Rendi Chevi, Radityo Eko Prasojo, and Alham Fikri Aji. Nix-tts: An incredibly lightweight end-to-end text-to-speech model via non end-to-end distillation, 2022.