**Linnéuniversitetet**
Institutionen för datavetenskap, fysik och matematik

Examensarbete

# Automatic Transcript Generator for Podcast Files

Andy Holst
2010-08-01
Ämne: Datavetenskap
Nivå: C
Institution: DFM, Institutionen för datavetenskap, fysik och matematik
Kurskod: 2DV40E

# Summery

In the modern world, the Internet has become a popular place for visitors to gather information, that can either exist in text or media form. For people with hearing problems, deaf people and for search engines, it is hard to take part of the content of the speech in the digital media files known as "podcast." To solve this problem; speech recognition can be used to generate a transcript of the podcast content; so search engines, deaf people and people with hearing problems can access his information.

The "Auto Transcript Generator" "ATG" application uses speech recognition technology to be able to transcript content from a podcast file, the ATG uses MPlayer to extract and convert podcast content if necessary and sends this audio file to the speech recognition system called Sphinx-4, it's the decoder that generates the transcript with the recognized words and writes it to a text file.

Speech recognition takes a PCM digital speech input and generates a "frequency domain" of the speech sounds and takes these features and compares it with the features in the grammar file to be able to recognize what words was spoken by converting the phonemes to word (phrases). The speech recognizer uses Hidden Markov Models "HMMs" for learning what phonemes forms specific kinds of words and HMMs to be able to recognize what kinds of phonemes match a specific word. The ASR "auto speech recognizer" uses a acoustic model, language model and a dictionary to be able to work. One of many speech recognizer systems is the Sphinx-3 and Sphinx-4.

The method of the tested ATG system is a quantitative method where the learned speech content with its transcript is being measured mainly of word accuracy in percentage from all of the total words. The reliability is stable to test word accuracy, and the same decoder results are expected with the same settings being run over and over, where the measuring instrument is the Sphinx-3 decoder

The acoustic model training is done with the SphinxTrain application, where 8 cases of acoustic models has been trained with different number of speakers. The decoder tests of the trained data are done with the Sphinx-3 decoder. The ATG system is finalized by coupling the ATG-handler Java file with MPlayer embedded commands, and with help of the Java Transcriber "speech recognizer" file based on the Sphinx-4 library.

The 8 different acoustic models was trained with SphinxTrain, the Sphinx-3 decoder results shows that the best case for word accuracy is 75 % with the biggest acoustic model and the worst case of word accuracy is 50 % if the speaker's accent differs from the trained speakers or if there is a lot of noise in the speech audio.

The theory of speech recognition coincides with how the Sphinx decoders works, simply by searching through the acoustic model for equivalent sounds compared to the input sound, and keeping track of the phonemes until a pause is reached, during this pause the decoder searches through the language model for equivalent series of phonemes that match a specific word.

The speech recognition is an interesting field, since there is definitely a demand to implement it in different kinds of applications to satisfy people with disabilities, search engines etc. I had no idea that training an acoustic model with 2452 speakers and with literally 60 hours of speech content would take 8 hours instead of 24 hours to train and give a stable word accuracy of 75 % of all of the words.

Creating a speech recognition system is a consuming task, it is fairly easy with the experts help to get decent accuracy at 75 % by assuming that the necessary time is taken into consideration to test and understand how optimizing can be done on the acoustic model, language model and the decoder.

One way to increase the word accuracy to 90 to 95 %; can be done with the LDA/MLLT acoustic model transformation and with out-of-vocabulary meaning (language model) enabled.

# Sammanfattning

Internet är ett mycket poplärt ställe för besökarna att hämta information ifrån, den här informationen kan existera i textfiler eller i media-filer "podcast". För besökare med hörsel problem eller besökare som är döva och för sökmotorer så är det svårt att ta del utav innehållet i media-filerna i form utav tal. För att råda bot på detta problem, så kan man använda sig utav taligenkänning-tekniken för att generera transkribering utav podcast-innehållet.

Auto Transcript Generator "ATG" applikationen använder sig utav taligenkänning-tekniken för att nedteckna innehållet i en podcast-fil, ATG-systemet använder sig utav MPlayer för att eventuellt behöva extrahera eller konvertera ljud-spåret och skicka detta ljud-innehåll till taligenkänning-systemet, Sphinx-4, där Sphinx-4 avkodaren genererar nedteckningen utav de matchade orden (fraser) och skriver det till en textfil.

Taligenkänningstekniken tar en eller flera PCM-digital-ljud och genererar en "frekvens domän" utav tal-ljuden och för dessa egenskaper jämför det med egenskaperna i grammatik-filen för att matcha ord genom att konvertera tal-ljuden till respektive ord. Moderna taligenkänningsystem använder sig utav *Hidden Markov Modeller* "HMMs" för att lära sig om hur talljud i en viss ordning tillsammans bildar ord och att kunna känna igen vilket ord en samling foneter motsvarar. Automatisk-taligenkänning-system som CMU "Carnegie Mellon University" Sphinx erbjuder, använder sig utav en akustik-modell, språk-modell och ett lexikon för att fungera. En av de populära taligenkänning-systemen är Sphinx-3 och Sphinx-4.

Metod för implementation utav ATG-systemet är kvantitativ där all tal-innehåll med dess transkribering mäts exakt hur varje ord i innehållet känns igen utifrån den tränade akustikmodellen, med vald lexikon och språk-modell. Tillförlitligheten är stabil, upprepade avkodningar utav tal-innehållet ger samma resultat med samma avkodningsinställningar.

Akustik-modell träningen är gjord med SphinxTrain-applikationen, där 8 fall utav akustikmodeller har tränats med ett antal olika talare och nedteckningar som är testade med "Sphinx-3"-avkodaren. ATG-systemet är färdigutvecklad genom att koppla på ATG-Handler-Java filen med inbyggda MPlayer-funktioner (media-funktioner) och med hjälp utav "Transcriber.java"-filen "taligenkänning-modulen" baserad på Sphinx-4 biblioteket.

8 olika akustik-modeller tränades med SphinxTrain, utifrån test resultaten med Sphinx-3 avkodaren så visade sig att akustik-modellen med flest talare och flest senoner (bundna-tillstånd) ger i bästa fall en tillförlitlighet på 75 % utav alla ord i tal-innehållet och i värsta fall 50 % tillförlitlighet om talarens dialekt skiljer sig kraftigt utifrån de tränade talarna eller om det är mycket brus i tal-innehållet.

Teorin om taligenkänning överensstämmer om hur Sphinx-avkodarna fungerar, genom att söka igenom akustik-modellen efter ekvivalenta tal-ljud utifrån inkommande tal-källa, och avkodaren håller koll på alla matchade foneter till talet slutar med en fonet-pause, under denna paus sökes språk-modellen igenom för likartade serier utav foneter som motsvarar ett specifikt ord.

Taligenkänning har många tillämpningsområden som kan hjälpa människor med olika behov till vardags men det viktigaste är nog sökmotorerna, som inte kan i nuläget söka igenom podcast-innehåll för indexering. Att träna en akustik-modell med 2452 talare med i princip 60 timmar tal-innehåll tog drygt 8 timmar istället för 24 timmar som ger en ord-tillförlitlighet på 75 % med Sphinx-3- och Sphinx-4-avkodaren i bästa fall.

Skapandet utav ett taligenkänningsystem är tidskrävande, men relativt enkelt att utföra med experternas hjälp för att få en hyfsad ord-tillförlitlighet på åtminstone 75 % förutsatt att man tar sig tiden att testa och förstå hur optimering utav tillförlitlighet kan utföras på akustik-modellen, avkodaren och språk-modellen. För att nå upp till ord-tillförlitlighet på 90 till 95 % krävs det att man har LDA/MLLT-transformation tränad på akustik-modellen och vokabulär som saknar betydelse inställt på språkmodellen.

# Abstract

In the modern world, Internet has become a popular place, people with speech hearing disabilities and search engines can't take part of speech content in podcast files. In order to solve the problem partially, the Sphinx decoders such as Sphinx-3, Sphinx-4 can be used to implement a Auto Transcript Generator application, by coupling already existing large acoustic model, language model and a existing dictionary, or by training your own large acoustic model, language model and creating your own dictionary to support continuous speaker independent speech recognition system.



**Keywords:** Auto Transcript Generator, embedded MPlayer media functions, Podcast, Speech, Speech recognition, Hidden Markov Models, Sphinx, Sphinx-3, Sphinx-4, Decoder, SphinxTrain, Trainer, Acoustic model, Tied-states, Senones, Language model, Dictionary, Transcript, Word accuracy, Word error rate

# Preface

The project to research about speech recognition came from own curiosity about how machines can recognize speech into words. This field has many applications, one of the most interesting ones is the service robots that is being researched and implemented today that can interact with humans through speech, here the country Japan is the pioneers when it comes to robot interaction through speech recognition, speech synthesizer and the field of semantics.

I want to thank all the guys at the CMU Sphinx IRC channel, cmusphinx on "irc.freenode.net" for all the questions I had about training an acoustic model, special thanks to Nickolay V. Shmyrev for all of his aid.

# Table of Contents

# 1 Introduction

## 1.1 Background

In the modern world, Internet has become a very popular place for people to gather information, this information can exist in many forms, the most well known are text, audio, video and pictures. Information in audio and video format without a transcript is hard to understand for people with hearing difficulties, people that has a hard time to decode speech and to deaf people and not the least, search engines, like google [1], if it can't be found, then it don't exist.

In the video section we have allots of video logs mainly from the hosting site, youtube [2], here, the voiced-audio can be auto transcribed on the fly with youtube's speech recognition technology "ASR" [3] if the author of the material has allowed it, this works pretty well as long the voice-sound is based on English. However, this does not mean that search engines can search for transcripts (containing words) in the video/audio files hosted on youtube, unless they are created by the youtube author, which can be created either manually or by sending the links or media file to today's company transcript services which costs money and is time consuming.

Podcast [4] is much more than video files, they can also be audio files, which many radio stations and blogging sites have, often these podcast's are missing transcripts as well so people with the difficulties I described above can't take part of this kind of information. The biggest problem, is that search engines can't search for words in podcast files, so if the author, who wants their content in the podcast files to be search able, then they need the transcript of the respective podcast hosted on the web as well to be sure that the podcast content exist on the web for search engines like the popular google.

A simple google search after "+podcast +auto +generate +transcript +app +program" Figure 1 doesn't provide any valid up to dated links about any kind of speech recognition product that can be downloaded and used to generate transcripts from podcast files. However, there are services like "Podcast Captioning" [5], "Transcript of audio" [6], but there is no free/trial podcast-to-transcript programs that you can download and try to generate your own transcript, they only exist for company's that are willing to pay for speech to transcript applications, often used by the Medical company's by their service department [7].

Transcripts that are created manually today, either by user or by a company is time consuming and requires the presence of the user. Consequently the study of automatic transcript generation on podcast files appears to be a valid choice of research.

## 1.2 Purpose

The purpose is to use one of the speech recognition technology that exist on today's development frameworks and develop an application to support an automatic transcript generator for a podcast file so deaf people and people with voice language understanding problem can take part of the transcript information related to the podcast. And lastly, so search engines can index each podcast content in greater detail related to their existing transcript hosted on the web.

## 1.3 Goal

The goal is to develop an Auto Transcript Generator application that takes a podcast file as input and send over the audio to the speech recognizer for processing, and finally save the recognized words to a text transcript file that is being sent over to the transcript smoother module for removing unnecessary repetitive words and finally save the smoothed up transcript file to a text file. The main purpose is that the speech recognizer should work as smooth and

accurate as possible, in order to achieve this, a trained acoustic model and language model is going to be used, preferably with a new trained acoustic model and language model that is speaker-independent but requires the language to be in English.

The ATG system's *speech recognizer accuracy* is going to be based on the decoder test result that was done with the selected trained acoustic model, selected dictionary and selected language model along with the speakers' speech audio files and their transcripts to the speech audio files.

### 1.3.1 Auto Transcript Generator

Is going to be Java based application and have MPlayer [8] and its sibling, mencoder embedded into the system.

#### 1.3.1.1 Audio process

The ATG "Auto Transcript Generator" takes the podcast file as input, and if necessary, depending on the podcast format, the audio extraction and eventually audio conversion needs to be done before sent over to the speech recognizer routine in pure wav "wave" format. In order to achieve this, the ATG-Java application is going to call for MPlayer and mencoder (sibling to MPlayer) commands in order to achieve necessary extraction and conversion and send the processed audio stream as pure wave format file to the speech recognizer module.

#### 1.3.1.2 Speech recognizer

The speech recognizer is based on the Sphinx-4 framework that is the state-of-the-art speech recognition system written entirely in Java [9]. It's going to use a trained acoustic model and language model for the input audio file in wave format to provide the recognized words and send them to the transcript smoother module.

#### 1.3.1.3 Transcript smoother

The recognized words from the speech recognizer is saved to a text file and smoothed up as much as possible by the transcript smoother to deal with removing unnecessary repeated words.

#### 1.3.1.4 Trained acoustic model and trained language model

The acoustic model and language model is going to be trained with the tools the CMU Sphinx team provides, otherwise is a trained language and trained acoustic model is going to be used instead, the first choice is preferable for the best results since the acoustic model can add more speech audio files for larger support of independent speakers to become even a better speaker-independent speech recognizer "decoder".

#### 1.3.1.5 A broad amount of podcast files

To get the best results, and broad amount of different types "themes" of podcast files on English is going to be tested to be able to draw the best conclusion from the results.

## 1.4 Limitations

I am going to limit my research on the Sphinx engines [10] and to train my acoustic model, language model with the trainer tools the CMU Sphinx team provides, and to use these two models with the Sphinx-4 [11] framework for the speech recognition on podcast files.

Sphinx-4 is based on the Java language, which the ATG application is going be built upon and entirely Java based, in order to be able to extract and convert the audio from almost any podcast file format. The MPlayer application is going to be embedded by the ATG application to make its internal MPlayer and mencoder command calls to fully be able to extract and convert the audio from the podcast (almost any type) input file if necessary.

# 2 Theory

## 2.1 Auto Transcript Generator application

The "Auto Transcript Generator" application has 7 defined modules "routines" to work properly Figure 2. There is 5 possible ways the application routine can take. However, if the podcast file is valid, it will always be sent in to the proper format, wave (wave) format to the speech recognizer that will send it to the transcript smoother and finally save the transcript file.

If the podcast file is in invalid format, then an exception will be thrown and the user will be informed about the problem.

If the podcast is a wave file, then it will be sent to the speech recognizer module that will generate the recognized words from the audio wave file and at completion it will send the generated transcript to the transcript smoother module that will smooth up unnecessary words and save the transcript file.

If the podcast file is not wave file and not a video file, then the audio file will be sent to the audio converter module and converted to wave format and then be sent to the speech recognizer.

However, if the podcast file is valid and is a video file, then the audio will be extracted from the video stream, if the extracted audio stream is in wave format, then it will be sent to speech recognizer, otherwise it will be sent to the audio converter module.

## 2.2 Audio process

### 2.2.1 Basics

Today's podcast files is either in a video format with a composed video track and an audio track or they can be in an audio format with specific audio track. There is 3 possibilities that can take place before the audio can be processed, if it is a video file, then the audio needs to be extracted, and if it's not in wave format, then it needs to be converted to wave. The other possibility is that it's a audio file that is not in wave format, then it needs to be converted as well before being processed. For this purpose, MPlayer is going to be used since it can extract and convert any podcast file to wave format that the speech recognizer, Sphinx-4, supports.

### 2.2.2 MPlayer

MPlayer is open-source, well maintained and updated with the latest media formats, it's easy embeddable in Java application using the slave mode [12] and it is ported for all the major operating systems (Windows, Unix, Mac) so the portability won't suffer so much by having MPlayer embedded into the ATG Java application. There exist an interesting tutorial on [13] to start with embedding the MPlayer commands into the Java application and is going to be used for the ATG system.

#### 2.2.2.1 Features

The features that MPlayer provides is long [14], it can stream literally any kind of media format from any place that you can think of and the documentation of the MPlayer application [15] gives you an insight of the very broad amount of possibilities you can do, so it is sufficient for what ATG application needs. It can easily extract and convert an audio of any podcast file, it takes about 1-2 seconds fully optimized to extract an audio from a podcast file containing about 45 minutes content and at the same time get it to raw wave audio format.

#### 2.2.2.2 Functionality

The MPlayer command-line program is very efficient to extract and convert audio files stored on the podcast file, all it takes is the command *"mplayer -quiet -ao pcm:fast:file=audio.wav -vo null -vc null -framedrop podcast"* where "audio.wav" is the output file and "podcast" the input file. The "-quiet" switch will make the extraction and conversion process allot faster since it does not have to output the results to the screen, the "-vo" and the "-vc" switches prohibits the possible existing video stream to be outputted to the screen, making the audio extraction and conversion to wave format at almost its full extraction and conversion performance. The only more performance gain the process can get if the output is being redirected to the "/dev/null" file which is possible in a Unix system. The audio.wav file is only overwritten or created if the podcast file contains a valid audio stream format, which is the main check for the exception.

## 2.3 Speech recognition

### 2.3.1 Basics

A high level overview of how speech recognition works is provided here [16], where the speech recognition's main responsibility is to transform the PCM digital audio into a better acoustic representation called "frequency domain", the next step is to apply a grammar so the speech recognizer knows what to expect and figure out which phonemes are spoken and finally convert the phonemes into words. The article *A Speech Recognition and Synthesis Tool* [17] provides a clear and concise overview of the speech recognition field and all the essential methods used in this process are exposed in *Statistical Methods for Speech Recognition* [18].

### 2.3.2 HMMs

*An HMM-based system, like all other speech recognition systems, functions by first learning the characteristics (or parameters) of a set of sound units, and then using what it has learned about the units to find the most probable sequence of sound units for a given speech signal. The process of learning about the sound units is called training. The process of using the knowledge acquired to deduce the most probable sequence of units in a given signal is called decoding, or simply recognition.*

This technology allows the system to get audio input transcribed or used to interact with a system [19]. A speech recognition system can handle either a unique speaker or an infinite number of speakers. Modern speech recognition system are based on HMMs, and Ghahramani [20] provides a thorough introduction to HMMs.

In the decoding process, every part of the speech signal gets transformed into features that gets scored against the acoustic model to generate the best matched sequence of phonemes. With the sequence of phonemes, a related search graph with HMMs states with a entry node gets generated, every branch from the entry node contains a sequence of related phonemes and possible paths generating final words at the common exit node called *senone*. The possible paths the search graph can take is literally endless from left to right transitions and with possible self transitions along the way, in order to determine the weights of the paths with its transitions, a language model is used to find out the best matched (most weighted) words from each branch in the search graph that has reached the exit node.

### 2.3.3 Acoustic model

Acoustic model is a file containing a statistical representation of distinct sounds that make up each word in the language model. There exist two types of acoustic models, the speaker dependent acoustic model and the speaker independent acoustic model. The speaker dependent

acoustic model has been designed to handle a unique speaker's speech, this acoustic model is usually trained from the concerned person. The speaker independent acoustic model is designed to handle speech from different people, especially the people who did not participate in training the acoustic model.

### 2.3.4 Acoustic model training

Training an acoustic model is done by transforming the speech signals into a sequence of vectors that represents certain characteristics of the speech signal "phoneme" and the parameters "features" are then estimated using the vectors (features) for each phoneme.

### 2.3.5 Lextree

An acoustic model with phonemes has a Lextree of all of the phonemes, where each phoneme in the language represent a tree with the specific phoneme as the parent node with one or several branches with combination of phoneme nodes where each leaf branch represents a senone "word". For large vocabulary, a large number of senones is needed.

### 2.3.6 Senones

One thing to remember about training acoustic model, is that a phoneme does not sound the same with different preceding and post-ceding phoneme neighbours when speech is being generated, in order to solve this problem, the triphone states are created generating more than 120 000 triphones because English contains about 50 phonemes. This number of senones is to large for modern computers to handle. To solve this memory space issue, clustering is done by cluster HMM states with triphones that share similarities, where each cluster is called a senone. The number of senones per Lextree cause the number of triphone states to be reduced greatly.

Senones provides improved recognition accuracy and pronunciation-optimization capability. Big number of senones gives better accuracy if the training data set is large. It's a matter of try and testing to go for optimal senones on the test data for best speech recognition accuracy.

### 2.3.7 Language model and Dictionary

A language model groups a broad list of words and their probability of occurrence in a given sequence. A dictionary list phonemes that is associated to every word. The distinct sounds is what the phonemes is made of and forms the word.

### 2.3.8 Decoder

The decoder takes sounds spoken by a user and searches the acoustic model for equivalent sounds. When a match is found, the decoder determines the phoneme that correspond to the sound and builds up the search graph with the related matched phonemes. During the decoder process, it keeps track of all found phonemes until the user's speech reaches a pause. During this pause it searches the dictionary file to map the phonemes to the matching words, and it uses the language model to determine the weights of the final state words. If matches is made, the decoder will return the best with most weighted (score) matching word or phrase to the calling program. In order to limit the search space for the decoder, the pruner is set to set the minimum weight path allowed, by setting this limit, the reduction of paths to final state words is reduced greatly. By definition, low cost phoneme transitions lower then the minimum weight is disregarded.

## 2.4 CMU Sphinx

The CMU Sphinx Team's tools is presented since they focus is speech recognition and it is the core part of the ATG system. The following explanations are based on the official website and documentations and tutorials provided by the team [21]. The Sphinx team provides four decoders, they are PocketSphinx ( used in live applications ), Sphinx-2 ( used in interactive applications ), Sphinx-3 (*state-of-the-art large vocabulary speech recognition system*) and Sphinx-4 (*state-of-the-art speech recognition system written entirely in the Java*).

*"CMU Sphinx is a collection of several incarnations of Sphinx, a versatile continuous speech recognition tool-kit from the Sphinx group at Carnegie Mellon University in Pittsburgh. It consists of two major kinds of components: trainers and decoders. The trainers (SphinxTrain and SimpleLM) are used to build acoustic and language models. These models are one input used by the various Sphinx decoders to transcribe digital audio. The decoders (Sphinx2, Sphinx3, and Sphinx4) perform the actual speech recognition. CMU Sphinx is versatile, in that it can be applied to small, medium, and large vocabulary speech recognition applications."* [22].

The Sphinx-4 framework will be used, since it is a versatile and flexible continuous speech recognition system and is at least as good as the Sphinx-3 decoder, if not better and faster in some cases compared to the recognition tests. Since it is Java-based, has good API support for web services, the object oriented is incorporated in the programming language which makes it easier to deal with, has good Java documentation and is well maintained and updated often.

In order to use the Sphinx tools in an optimal way, some software are needed, Perl to run SphinxTrain scripts, C compiler to compile Sphinx sources, the Java platform and Apache Ant for using Sphinx-4.

## 2.5 Sphinx-4

Sphinx-4 is an open source project led by Carnegie Mellon University, Sun MicroSystems Inc., Applied Computer Science Group - University of Bielefeld, Mitsubishi Electric Research Laboratories. A white paper exist that presents an overview of the framework [23]. First of, Sphinx-4 is modular and pluggable framework that incorporates design patterns from existing systems, with sufficient flexibility to support emerging areas of research interest. It supports any acoustic model structure and supports most of the language models.

### 2.5.1 Framework

The Sphinx-4 has three principal modules, the *FrontEnd*, the *Decoder* and the *Linguist* getting material of the *Knowledge Base*.

The FrontEnd gets a single or several input signals and computes them so that a sequence of *Features* (vectors) is created.

The Linguist generates a *SearchGraph* by translating any kind of standard language, with the aid of pronunciation information contained in a *Lexicon*, and with the structural information stored in sets of *AcousticModel*.

The *SearchManager* component located in the Decoder uses the Features from the FrontEnd and the SearchGraph from the Linguist to perform the actual decoding, generating *Results*. During or prior to the recognition process, the application can issue *Controls* to each of the modules.

The *ConfigurationManager* allows the system to be configurable with the number of parameters it provides, it also supports for being flexible by allowing the system to dynamically load and configure modules at run time. Multiple tools are provided by the framework to be able to track decoder statistics such as word error rate, run time speed and memory usage. *Utilities* are also provided that supports application-level processing of recognition results, such as

obtaining result lattices, confidence scores and natural language understanding.

### 2.5.2 Features

The features of the speech recognition system, Sphinx-4, can be configured from the XML configuration file, due to that, the Java code to run the system is initially brief.

# 3 Method

The ATG (Auto Transcript Generator) for podcast files is implemented in the Java programming language since the speech recognition framework, Sphinx-4 is Java based. To describe the ATG-system briefly, it will take 3 arguments, one for the transcript output file, one for the wave file (created if necessary) and one for the podcast input file. The ATG system will be able to extract and convert the podcast audio track to a wave file if necessary.

The audio track in wave format will be the input for the speech recognizer decoder that will generate a transcript file from the podcast content, and lastly, to smooth up the transcript file of unnecessary words defined for the English language.

During the implementation process, an acoustic model and a language model is going to be trained with the tools that the CMU Sphinx team provides to get the best results from the speaker independent continuous speech recognizer decoder "Sphinx-4 framework" in the ATG system.

The trained acoustic models is going to be tested against their own transcripts in order to determine the best speech recognition accuracy for future speech recognition use on different media content with independent speakers (podcast).

## 3.1 Quantitative method

It is going to be a truly a quantitative method, where the time to run and process audio is going to be measured and the results is going to be measured for how reliable and correct they are compared to existing transcripts.

## 3.2 Selection

In order to meet this goal, an acoustic model is going to be trained with big number of English speakers and their respective transcripts from the VoxForge website [24] and by using the SphinxTrain [25] application. For the speech recognition tests, the primary decoder is going to be the Sphinx-3 since it is 3 to 5 times faster then the Sphinx-4 decoder, but after successful decoder tests, the Sphinx-4 decoder can be setup with the same settings as the Sphinx-3 decoder to get at least the same results but cost of the time factor.

### 3.2.1 Representative the reality

All of the speakers with their respective transcript from the English speech corpus with 8k samples and 16 bit sample audio files is being counted for. It is fair to assume that the accent between the speakers will differ greatly, and therefore be able to support large number of independent English speakers and the acoustic model can be classified as independent speech interpretation for the speech decoders such as Sphinx-3 and Sphinx-4 with good accuracy results of the testing data.

## 3.3 Reliability, validity and objectivity

### 3.3.1 Critical and creatively thinking

The Sphinx-4 framework is a versatile and trustworthy modern speech recognizer decoder and has the ability to test out many of the different settings, however, the settings will be critical set so a large vocabulary of words can be recognized by different speakers so the speaker independent environment is met. The creativity is endless, you can literally setup any kind of application to use this technology to generate transcripts, command a computer or even a

robot by voice, you can literally build on this technology and build some kind of AI to let the computer respond with speech synthesis on your inputs (arguments) etc.

The decision to create an auto transcript generator is one of many applications that I feel is necessary today, especially if you do lots of video blogs and wants to let every one be able to take part of the content.

The sources for acoustic training and speech recognition will be done from tutorial sources that the Sphinx team provides and the accuracy of all the words from the transcript will be measured and the accuracy percentage by number of speakers and senones with large data set (speakers) is the key features to show how well the speech recognition system works in best case and in worst case.

### 3.3.2 Validity

The intentional data of the speech such as word accuracy and word error rate of all the words was measured that indicates how well the speech recognition system works in general use in best case and in worst case.

#### 3.3.2.1 Research

The VoxForge English speech corpus satisfied the intended speech recognition data results and indicated how well the speech recognition will act in best case and worst case depending on the current decoder setup and with the trained "measured" acoustic model, language model and the dictionary that is easy to integrate in the decoder system. By carefully studying the SphinxTrain and Sphinx-3 decoder tutorial, you know what results to expect and how you can improve the accuracy.

### 3.3.3 Measuring instrument

The measuring decoders such as Sphinx-3 and Sphinx-4 will give the same accuracy if they have same decoder settings and the results from the log files they generate indicates how well the speech recognition system works with the chosen acoustic model, language model and dictionary with any provided English speech in best case and in worst case compared from the decoder results from the trained acoustic model.

## 3.4 Reliability

The reliability of getting the same measured data results is very likely on the same speech corpus material that is being analysed for speech recognition, it depends how well the acoustic model and the language model has been trained, the more speakers that is trained, the better and more speaker independent reliability it will result into, cause of the speakers unique accent to say words differently.

### 3.4.1 Measuring instrument

The Sphinx-3 and Sphinx-4 decoder gives the same results of the same speech data as input, if you however get lower accuracy then the best and worst expected word accuracy of all the words from the tested trained acoustic model, language model, then you know that the decoder is not setup properly.

### 3.4.2 Objectivity

The objectivity was crystal clear from start one, it is very good thing if search engines and people with hearing difficulties can take part of the detailed podcast content "speech" that is on the Internet today, and be able to do so with a reliable speech recognition system that is easy to use. It is the reliable and generated transcript results that matters in feasible time.

## 3.5 Implementation

### 3.5.1 Measuring instrument

For training the acoustic models, the SphinxTrain application is going to be used, the language model and the dictionary is provided in the voxforge-en-r0_1_3.tar.gz brought from VoxForge site. For the speech recognition tests on the trained acoustic models the Sphinx-3 decoder is used mainly cause of the speed factor of 3 to 5 times faster then the Sphinx-4 decoder.

### 3.5.2 Preparations

Before the project of training an acoustic model and a language model for English speech recognition, I had basic understanding to get speech audio and transcripts from VoxForge and that the CMU Sphinx Team provides with the necessary tools to be able to train an acoustic model and language model and to test the trained acoustic model and language model against one of the Sphinx decoders that supports it.

### 3.5.3 Preliminary Investigation

Preliminary investigation shows that MPlayer application and the Sphinx-4 decoder, along with the CMU Sphinx team's training tools is sufficient for creating the ATG system. The CMU Sphinx team provides the necessary training tools and tutorials to setup both the acoustic model, the language model and to setup the Sphinx-4 framework [26].

### 3.5.4 System overview

We have already described the early stage flow chart of the experimental ATG system Figure 2, the implementation on the ATG system is going to be Java based, mainly because the Sphinx-4 speech system is Java based.

### 3.5.5 General system requirements

In order to start developing the ATG-system, some libs and applications needed were installed, the Java run time environment "JRE" [27] to be able to run Java applications, Java development kit "JDK" to develop Java applications.

On the audio extraction and conversion side, the MPlayer [28] application was installed, and in order to be able to make it smooth to compile and distribute both Sphinx-4 framework with the ATG system, the Apache Ant [29] binary distribution was installed along with Sphinx-4 framework system.

For the training tools, SphinxTrain [30] to train the acoustic model was installed and the "CMU SLM Tool-kit" [31] to train the language model might be needed.

The development system needs the GNU C compiler and Perl in order to use the training tools properly and to compile the MPlayer source code which is recommended, so it was made sure to be installed on the Linux system.

The main importance for starters before starting to developing Java applications for the Sphinx-4 framework and to compile Sphinx-4 Java code, is that the operating system that is

being used for development, must have *ANT_HOME* environment variable set to the directory where Ant is installed and the path to the Ant's bin directory, and that the *JAVA_HOME* environment variable must be set to the latest version of the JDK installed, and this was done as shown below on a Unix/Linux system in bash:

```
1  export ANT_HOME="/usr/share/ant/"
2  export JAVA_HOME="/usr/lib/jvm/java-6-sun-1.6.0.19/"
3  export PATH=${PATH}:${ANT_HOME}/bin
```

### 3.5.6 Audio conversion and audio extraction

The two modules shown in the Figure 2 and processed in combined module with the help from the MPlayer application that is embedded into the ATG system.

MPlayer section in the theory part described the command process of extracting and converting the audio track from the podcast if necessary, that is, the MPlayer command *"mplayer -quiet -ao pcm:fast:file=audio.wav -vo null -vc null -framedrop podcast"*. The podcast is the input and the "audio.wav" is the output, the other switches are just to null video output and mute unnecessary audio output to speed up the audio conversion and audio extraction process. The "audio.wav" is only created if there exist a valid audio format track on the podcast file and that it's not equal to raw wave (wav) audio format.

However, it is necessary for the MPlayer to be able to identify the tracks that is on the original podcast file, and take the right decision if audio extraction and conversion is necessary, this is going to be done with the command *mplayer -identify -frames 0 podcast*, where the -identify switch identify outputs of all recognizable tracks formats and settings, and the next step of what to do is based on this information logged to a stream.

There is basically two cases that is taken into consideration before what to do, one, if the podcast contains a video track and an audio track, then an extraction and dump or conversion of the audio track is necessary; two, if it is an audio podcast file, then check format audio type and convert it only if its not a raw wave format. It is necessary that audio extraction and conversion takes place if audio track is not a wave format, this is necessary cause Sphinx-4 only supports raw audio and wave audio formats.

These two cases are easily achievable with the two commands described above in this section, however, the MPlayer standard output and the standard error output is going to redirected to the same stream in order to handle the right decisions, see Listing 1. With help of the *LineRedirecter* class, it is possible to output the standard output and error output to the same stream and to let the MPlayer stay idle in the background, so you can redirect input streams to the MPlayer process and get back the default output stream directed to the *BufferedReader mplayerOutErr*, see Listing 2.

With the code above and the predefined settings, it is possible to take the right actions to identify if necessary audio extraction and audio conversion is needed or if just audio conversion is necessary. If not, then the podcast file can be used at input for the speech recognizer instead of a new created wave file.

### 3.5.7 Speech recognition

The main purpose for why the Sphinx-4 framework has been chosen is because it has proven to be reliable speech recognition system after two decades of development. It is also the latest speech recognition framework that CMU Sphinx team provides and is maintained frequently. The big factors to use it is the big ease of use to work with it (configurations) compared to other speech recognition decoders, and lastly it is a versatile and flexible continuous speech recognition

system and highly flexible modular architecture system, see the architecture Figure 3. The features the Sphinx4 framework provides is going to be taken advantage of to satisfy the needs of the ATG system.

In order for the speech recognition system to be able to work as a complete state-of-the-art HMM-based speech recognition system, the framework (Sphinx-3/Sphinx-4) needs the trained acoustic model, language model, dictionary and the filler dictionary.

### 3.5.8 Acoustic models training

#### 3.5.8.1 SphinxTrain preparation

In these tests, the SphinxTrain was used to train the acoustic models for the small and big training data.

The Sphinx-3 decoder was used to retrieve the SER "sentence error rate" and the WER "word error rate".

For starters, before the acoustic models got trained, all the necessary steps how to proceed with basic acoustic model training was read from the CMU Sphinx's SphinxTrain tutorial.

To be able to train the acoustic models, all the 8k samples, 16 bit sample audio data with their respective transcripts was downloaded from the VoxForge website containing about 3,2 GB compressed English data. The *voxforge-en-r0_1_3* SphinxTrain setup was used as a guideline and baseline for my acoustic model training setup.

The voxforge-en-r0_1_3 SphinxTrain setup includes a dictionary, filler, phone, and a language model for the VoxForge acoustic 8k samples, 16 bit sample English speech corpus audio data.

The dictionary file *voxforge_en.dic* contains over 100,000 of words with their respective phonemes to support the most known common English words that also covers all the words from the transcripts provided with the speech data, this is a dependency to get the accuracy level at a decent level (fifty % correct of all the words for large vocabulary data with short utterances per transcript line.)

The language model, voxforge_en.lm is basically generated with help from CMU-Cam_-Toolkit_v2 application *"Statistical Language Modelling Tool-kit"* based on all the 41924 transcript sentences; this language model creation is done from the transcription that belongs to the VoxForge 8k samples, 16 bit sample speech corpus.

The SphinxTrain includes a filler file *voxforge_en.filler* for filler sounds such as <s>, <sil>, and </s>. The *voxforge_en.phone* includes the phones that is used by the training set and even phones not used by the training set. The voxforge_en.transcription contains the transcription of each audio file surrounded by the markers <s>, and </s>. The feat.params and sphinx_train.cfg is generated by the SphinxTrain setup.

Before the acoustic models got created, the SphinxTrain was checked out in the cmusphinx's trunk folder in the Linux environment system with a subversion checkout and installed with the following commands:

```
1  svn co https://cmusphinx.svn.sourceforge.net:/svnroot/cmusphinx/trunk/
       SphinxTrain
2  cd SphinxTrain
3  ./configure
4  make
```

After the SphinxTrain installation, in the cmusphinx's trunk directory the folder voxforge_en was created and configured with the commands:

```
1  mkdir voxforge_en
```

```
cd voxforge_en
perl ../SphinxTrain/scripts_pl/setup_SphinxTrain.pl -task voxforge_en
```

Inside the voxforge_en directory, the files from the voxforge-en-r0_1_3, that is voxforge_en.dic, voxforge_en.filler, voxforge_en.lm (language model), and its bash scripts in the script folder was copied over to the top directory of the voxforge_en folder. There after the acoustic models was trained in eight different levels, in general the first 7 was set with training a small data set with just different senones and different number of speakers with their respective transcripts. Case 8 was trained with all possible speakers and their available transcripts and audio speech files for a very large vocabulary.

#### 3.5.8.2 Data preparation

For each case, the build.sh script Listing 3 was executed in the top level of the voxforge_en directory with the command:

```
bash scripts/Build.sh $NUM_SPEAKERS < speakers_list
```

The build.sh script adds the number of speakers in alphabetic order from the speaker list as input. The speakers with the proper transcript file named "PROMPTS" will be added to the etc/allprompts for further text processing their available transcripts and audio speech files for a very large vocabulary. This will result into that transcription text and audio text files will be created which are; the etc/voxforge_en.transcription containing all the transcription and the related audio files, the etc/voxforge_en.transcription.train for the acoustic training, the etc/voxforge_en.transcription.test for the decoding performance testing, the etc/voxforge_en.fileids.train for the acoustic training audio files and the etc/voxforge_en.fileids.test for the audio files for the decoding performance test.

The features of the wave files for both the training set and the decoding set will be generated in form of mfc extension files inside the feat folder at the top directory of voxforge_en folder.

#### 3.5.8.3 Train cases

**First case** First case, with 50 speakers (unique and non unique speakers accounted for) in alphabetic order with 426 transcript sentences and the training settings was set to 25 senones, 5 states per HMM with skip states enabled for continuous speech.

The "$CFG LTSOOV" set to 'yes', "$CFG DIAGFULL" set to 'yes', "$CFG WAVFILE EXTENSION" set to wave, "$CFG WAVFILE TYPE" set to 'mswav', the "$CFG LISTOFFILES" was set to "voxforge en.fileids.train" and the "$CFG TRANSCRIPTFILE" was set to voxforge_en.transcription.train. The other settings was standard generated by the SphinxTrain setup Listing 4. The estimated total hours training, 0.39 hours.

**Second case** Second case, with 50 speakers in alphabetic order and the training settings was set to 50 senones, the rest was the same as for the first case. Estimated total hours training, 0.39 hours.

**Third case** Third case with 100 speakers and 100 senones and 882 transcript sentences. The rest of the settings was same as for Listing 4. The estimated training time 0.38 hours.

**Fourth case** Fourth case with 100 speakers and 200 senones, but the most of the "sphinx train.cfg" file was setup like the "voxforge-en-r0 1 3" SphinxTrain setup. The "$CFG FEAT WINDOW" set to 0, the "$CFG FORCEALIGN" set to yes and states per HMM set to 3

and skip states disabled, the "$CFG FINAL NUM DENSITIES" set to 16 instead of 4 for continuous acoustic models.

The reason why this was tested instead of the default settings, was that increasing senones bigger then the speakers number seems to increase SER percentage, but it didn't in this case, most likely cause final states of matched words got reached (skip states disabled) and more senones was used, as you will see in the "Sphinx-3" decoding tests. The estimated training time, 0.38 hours, actually took 0.28 hours.

**Fifth case** Fifth case with 500 speakers and 4468 transcript sentences with 50 senones and the same training settings as for the first case. The estimated training time, 2.77 hours. The actual time it took to train the acoustic model, 34 minutes.

**Sixth case** Sixth case with 500 speakers and 500 senones, and the same settings as for the fifth case. Estimated training time 2.77 hours, actual training time, 34 minutes.

**Seventh case** 7th case with 500 speakers and 1000 senones, and the same settings as for the fifth case. Estimated training time 2.77 hours, actual training time, 40 minutes.

**Eighth case** 8th case with 2452 speakers and 3000 senones, the rest of the settings are the same as for the fourth case, as for the voxforge-en-r0_1_3 SphinxTrain setup. Estimated training time, 24.60 hours, actual training time, 6 hours and 4 minutes.

#### 3.5.8.4 Acoustic training process

For each case, after the data preparation was done with the execution of the bash script, build.sh in the top level directory of the voxforge_en, the next step was to do the acoustic training started with the command:

```
1  perl scripts_pl/RunAll.pl
```

### 3.5.9 Speech recognition tests

In this section, the eighth trained acoustic models are being tested with the Sphinx-3 decoder. The intention from the start was to test them out with the Sphinx-4 decoder, but, since Sphinx-3 decoder is three to five times faster then the Sphinx-4 decoder and easier to start out with, since Sphinx-3 is easy to setup and use the parameters from the SphinxTrain's config file for recognition.

The results from the Sphinx-3 decoder should be the same with Sphinx-4 decoder if it's configured the same way as the Sphinx-3 decoder was configured. Sphinx-4 decoder is a Java port from the C based Sphinx-3 decoder and works at least as good as Sphinx-3 if not better, but in cost of the lesser speed factor.

#### 3.5.9.1 Sphinx-3 setup

First, sphinxbase was checked out through subversion, since Sphinx-3 decoder requires the sphinxbase to be installed. There after, the Sphinx-3 was checked out in the cmusphinx's trunk folder. Sphinxbase was installed, then Sphinx-3 was installed, the commands are:

```
svn co https://cmusphinx.svn.sourceforge.net/svnroot/cmusphinx/trunk/
    sphinxbase
svn co https://cmusphinx.svn.sourceforge.net/svnroot/cmusphinx/trunk/sphinx3
cd sphinxbase
./autogen.sh
./configure
make
cd ../sphinx3
./autogen.sh
./configure --prefix=`pwd`/build --with-sphinxbase=`pwd`/../sphinxbase
make
make install
```

After this, the Sphinx-3 was attached to the voxforge_en folder with the commands:

```
perl ../sphinx3/scripts/setup_sphinx3.pl -task voxforge_en
```

where the *sphinx_decoder.cfg* Listing 5 got generated inside the etc folder in the voxforge_en top directory

#### 3.5.9.2 Recognition evaluation measures

Nickolay V. Shmyrev speech recognition developer states that the Sphinx decoders gives an accuracy of 50 % WER at decent level with large vocabulary data set with small utterances [32] and with default trained acoustic model with SphinxTrain. However, if there is audio noise or if the accent differs from the trained data, then expect the WER on the tested trained acoustic model to be a multiple of a factor 2 related from the speech recognition performance test on the original trained speakers.

In order to efficiently evaluate the Sphinx-3 decoder (Sphinx-4 decoder too), output text called hypothesis is aligned with the actual transcription named reference. Three error types are distinguished in the speech recognition process. First of is the substitution that deals with the words that are wrongly recognized. Secondly, the insertion deals with additional in the hypothesis that is irrelevant to the reference "transcript sentence". And thirdly, the deletion is counted for the words that is present in the reference, but not in the hypothesis.

Two measures permit to quantify the errors, namely *Word Accuracy* and *Word Error Rate*, they are determined as:

$$\text{WA} = \frac{\text{total words - substitutions - deletions - insertions}}{\text{total words}} \tag{1}$$

$$\text{WER} = \frac{\text{total word errors}}{\text{total words}} \tag{2}$$

$$\text{total word errors} = \text{substitutions} + \text{insertions} + \text{deletions} \tag{3}$$

where *total words* is the number of words in the reference transcript.

#### 3.5.9.3 Sphinx-3 decode process

For each trained acoustic model, the decoding test for each case was executed when the proper transcripts like the fileids.test, transcription.test and acoustic model was set through the sphinx_decode.cfg file in the top directory of the voxforge_en. The decoder accuracy execution was done with the command:

```
perl scripts_pl/decode/slave.pl
```

### 3.5.10 Finalize the ATG-system

The Sphinx-4 decoder contains allots of source Java files and XML configurations, among the Java files there exist a file called Transcriber.java and its respective XML configuration file. The Transcriber got copied over and modified so it takes two arguments, one for the audio wave file as input and the transcription file argument for where the recognized words gets written to. The new package "namespace" for the Transcriber.java file was edited to not be able to collide with the Sphinx-4's Transcriber.java file when the Sphinx-4 library is included.

The XML config file that Transcriber refers to got copied over and edited so the same decoder settings is used that the Nickolay V. Shmyrev used but with one modification, with the trained acoustic model, case 8, with literally 60 hours speech from 2452 speakers. The rest was finalized by letting the ATG's Java main method to call the ATG-handler Java file that calls for the MPlayer commands and the Transcriber methods if the three arguments, wave file as output (if necessary), transcription file as output and the podcast file are valid.

### 3.5.11 Demonstration of the ATG-system

Andy Holst demonstrating the ATG-system [33].

## 3.6 Criticism to chosen method

### 3.6.1 Majority group of observers

There is criticism to the chosen method is that case 8 could have the acoustic model trained with the implemented rejection of OOV ( out-of-vocabulary ) speech and with the MLLT transform parameter enabled and calculated to reach the 90 % word accuracy, that Nickolay V. Shmyrev used [34] with the same amount of data set (speakers and transcripts) but with the Sphinx-4 decoder instead of Sphinx-3 decoder.

The LDA/MLLT transform calculated which I never got to work by following the tutorial to the letter [35], increases the word accuracy with 25 %, which is roughly:

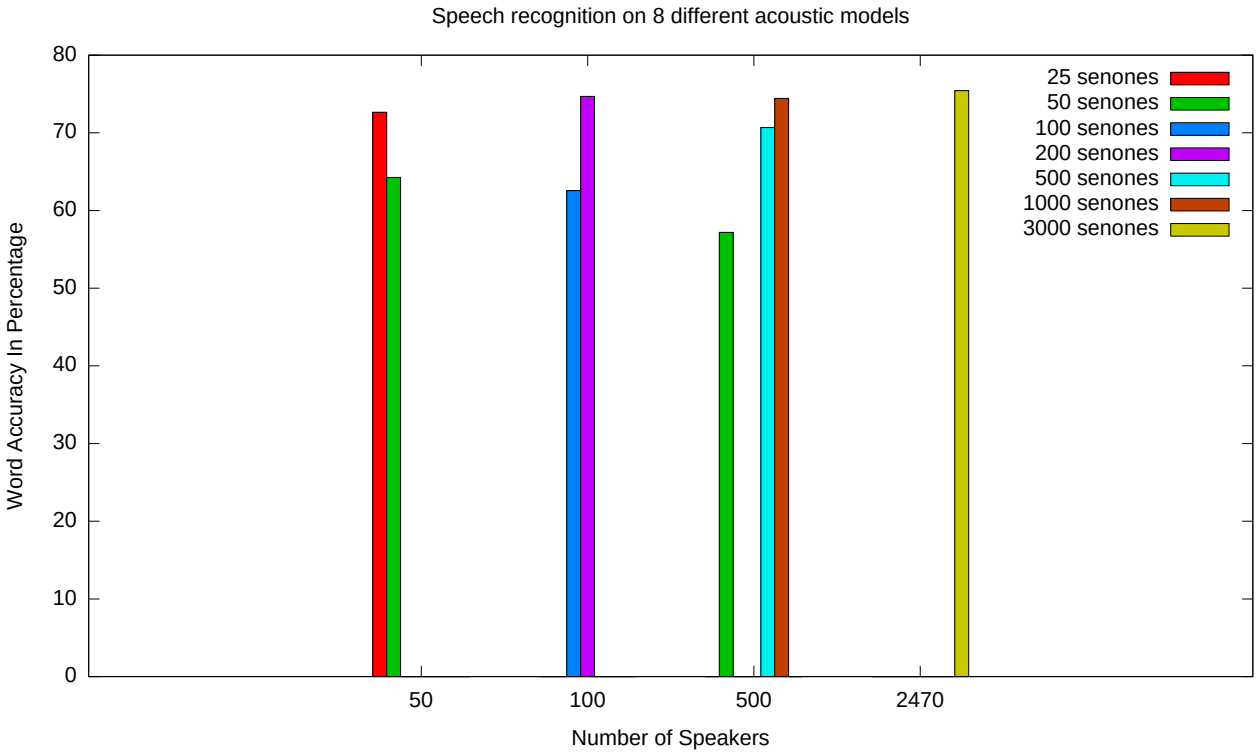
$$\text{WA'} = \text{WA} + (100 - \text{WA}) * 0.25 \tag{4}$$

75 % Result 1 WA for case 8 is good enough results for the first trial and test run with the experts advices of what the parameters should be to support large vocabulary independent continuous speech recognition. However, the acoustic model can easily be trained with the LDA/MLLT transform with the expense of the long acoustic training process time, either the estimated 24 hours training time or less depending on the machine hardware in use.

# 4 Results

From the results Result 1 and Table 1, in all the 8 cases, the speech recognition is at least 60 %. However, there is a relationship between the number of the trained speakers, vocabulary set and the number of senones.

The SphinxTrain tutorial insist that for big training data with many speakers and words, a large number of senones should be used and 3000 senones is recommended. 3000 senones is used in case 8 and that a bigger number of final densities is set to 16 densities instead of 4 for the 7 other small acoustic models.




**Result 1:** Sphinx-3 decoder results from the acoustic training.

**Table 1:** Speech recognition results from the trained data

| Case | Speakers | Senones | SER | WER | WA | Total words | Total sent. | Dec time |
|---|---|---|---|---|---|---|---|---|
| 1 | 50 | 25 | 57.4 % | 27.34 % | 72.66 % | 428 | 47 | 3 min |
| 2 | 50 | 50 | 48.9 % | 35.75 % | 64.25 % | 428 | 47 | 3 min |
| 3 | 100 | 100 | 65.4 % | 37.43 % | 62.57 % | 7869 | 882 | 5 min |
| 4 | 100 | 200 | 52.9 % | 25.31 % | 74.69 % | 7869 | 882 | 15 min |
| 5 | 500 | 50 | 67.3 % | 42.82 % | 57.18 % | 4468 | 496 | 10 min |
| 6 | 500 | 500 | 57.5 % | 29.32 % | 70.68 % | 4468 | 496 | 12 min |
| 7 | 500 | 1000 | 55.2 % | 25.56 % | 74.44 % | 4468 | 496 | 12 min |
| 8 | 2470 | 3000 | 55.0 % | 24.57 % | 75.43 % | 354088 | 37732 | 22 h |

# 5 Analysis

The theory of speech recognition coincides with how the Sphinx decoders works that is part of the ATG system. Simply by searching through the acoustic model for equivalent sounds compared to the input sound, and keeping track of the phonemes until a pause a reached, during this pause the decoder searches through the dictionary model for equivalent series of phonemes that map matching words. Finally the language model defines which matching word with the highest score is returned to the calling program.

The number of senones was relative okay compared to the number of vocabularies, for case 8, most likely a bigger number of senones could have been used to generate better speech recognition.

All of these steps worked exactly as processed in the recognition "decoder" tests, so you know what the speech recognition accuracy to expect in best case and in worst case from the other speaker data compared to the trained speaker data.

# 6 Discussion

The speech recognition system is an interesting field. I had no idea that it would only take 8 hours to train the acoustic model with 2452 speakers that should take at lest 24 hours, I was very skeptic during the time that the recognition accuracy should be even so high such s 75 % WA.

The limitations was that LDA/MLLT transform calculation never worked as expected, that parameter would increase the WA up with 25 %, so instead of 75 % WA, it would be 81 % WA and that time was not enough to train more large acoustic models with more then 3000 senones.

Training a acoustic model is very time consuming task to aim for the highest WA and to setup the Sphinx decoder to use the best search space related to time space to get the best WA of the incoming audio within reasonable time.

# 7 Conclusion

The conclusion of creating a speech recognition system with good word accuracy is that it is time consuming, but doable if enough patience is counted for with going through the Sphinx training and decoder tutorials, and by following the recommended trainer and decoder settings by the expert recommendations.

Lastly in order to improve WA from base settings, if enough reasonable time is given , getting a word accuracy between 90 % to 95 % works for all speaker data by setting up the optimal search space, time space, out-of-vocabulary implemented and with noise filter implemented for the Sphinx decoder of choice if the base word accuracy is at least 80 %.

With the current word accuracy of 75 % for acoustic model with 2452 trained speakers is the best case, if the speaker that tests the speech recognition system with either different accent or with lots of noise the in the audio, then the WER will be by a factor two, resulting into that the WA will be 50 % in worst case. With this amount WA for the worst case, it is not sufficient to be counted for people with speech hearing disabilities nor the search engine, this can be only useful if the best WA is at 90 % or greater.

The goals got achieved to implement a ATG system that uses MPlayer for audio process, Sphinx-4 for speech recognition. The best case and worst case for WA was determined for the acoustic model case 8 with its dictionary and language model for all kinds of English speakers. The rest with the decoder can always be optimized when it comes to search space of matching words by configuring the Sphinx-4 decoder through XML settings.

# 8 References


[1] Google Inc. 2010, *Google*, Google search engine, 11th April 2010, <http://www.google.com>.

[2] Google Inc. 2010, *Youtube*, Youtube hosting media files "video mainly", 10th April 2010, <http://www.youtube.com>.

[3] Ken Harrenstien 2009, *Automatic captions in YouTube*, Describes about the possibility of automatic captions on video files hosted on youtube, 9th April 2010, <http://googleblog.blogspot.com/2009/11/automatic-captions-in-youtube.html>.

[4] Wikipedia 2010, *Podcast*, Podcast definition, 11th April 2010, <http://en.wikipedia.org/wiki/Podcast>.

[5] Automatic Sync Technologies 2010, *Podcast Captioning*, AST's CaptionSync automated web-based service delivers captions via email, within minutes of your media file and transcript upload. If you don't have a transcript, our integrated transcription service can get one for you, 10th April 2010, <http://www.automaticsync.com/captionsync/services/captions-subtitles/podcast-captions-2/>.

[6] Syntax Trans Inc. 2010, *Transcript of audio*, Offers speech to transcript services for medical companies, 9th April 2010, <http://www.syntaxtrans.com/>; <http://www.youtube.com/watch?v=vPgIoudZyeg>.

[7] Nuance Communication 2010, *Speech magic*, Is a speech to transcript system used by medical company's service department. 9th April 2010, <http://www.nuance.co.uk/speechmagic/>.

[8] MPlayer Team 2010, *MPlayer*, Mplayer is a sophisticated application and is more then just a advanced multi-media player, it can be used for many things, 10th April 2010, <http://www.mplayerhq.hu/>.

[9] Oracle Inc. 2010, *Java*, Wikipedia on the Java programming language, 12th April 2010, <http://en.wikipedia.org/wiki/Java_(programming_language)>.

[10] Carnegie Mellon University 2010, *CMU SPHINX*, CMU. has built a Sphinx engine to be able to deliver speech recognition frameworks to support speech to text, 10th April 2010, <http://cmusphinx.sourceforge.net/>.

[11] Carnegie Mellon University 2010, *Sphinx-4*, The Sphinx-4 framework is a speech recognizer system that allows to be equipped with trained language models to support different kind of speech recognition accuracy and performance, 10th April 2010, <http://cmusphinx.sourceforge.net/sphinx4/>.

[12] Mplayer Team 2010, *SLAVE MODE PROTOCOL*, Documentation of MPlayer's slave-mode protocol, 12th April 2010, <http://www.mplayerhq.hu/DOCS/tech/slave.txt>.

[13] Adrian 2010, *JMPlayer*, A tutorial how to embed MPlayer into a Java application, 13th April 2010, <http://beradrian.wordpress.com/2008/01/30/jmplayer/>.

[14] MPlayer Team 2010, *MPlayer Features*, A long list of MPlayer features, 14th April 2010, <http://www.mplayerhq.hu/design7/info.html>.

[15] MPlayer Team 2010, *MPlayer manual*, Documentation how to use the MPlayer application "mplayer and mencoder", 14th April 2010, <http://www.mplayerhq.hu/DOCS/man/en/mplayer.1.txt>.

[16] Engineered Station 2001, *How Speech Recognition Works* An high overview theory of how speech recognition works, 15th April 2010, <http://project.uet.itgo.com/speech.htm>.

[17] H. ElAarag and L. Schindler 2006, *A speech recognition and synthesis tool*, ACM-SE06, The University of Mississippi.

[18] F. Jelinek 1999, *Statistical Methods for Speech Recognition*, MIT Press, Massachusetts Institute of Technology.

[19] cicheung2008 2008, *DIY Voice Control Tank*, A demonstration of CMU Sphinx engine in action, 15th April 2010, <http://www.youtube.com/watch?v=f4LUBX6mwBk>.

[20] Z. Ghahramani 2001, *An Introduction to Hidden Markov Models and Bayesian Networks* International Journal of Pattern Recognition and Artificial Intelligence, World Scientific.

[21] Carnegie Mellon University 2008, *Robust group's Open Source Tutorial*, Demonstrates how to train the HMM-based speech recognition system, 16th April 2010, <http://www.speech.cs.cmu.edu/sphinx/tutorial.html>.

[22] sourceforge 2006, *CMU Sphinx* Project description of CMU Sphinx that is a speech recognition engine system, 16th April 2010, <http://sourceforge.net/potm/potm-2006-03.php>.

[23] Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf and Joe Woelfel 2004, *Sphinx-4: A Flexible Open Source Framework for Speech Recognition*, SMLI TR2004-0811, Sun Microsystems Inc, <http://cmusphinx.sourceforge.net/sphinx4/doc/Sphinx4Whitepaper.pdf>.

[24] VoxForge 2010, *Welcome*, VoxForge site that provides transcripts and audio data under the GPL license for speech recognition systems, 8th May 2010, <http://www.voxforge.org/>.

[25] Carnegie Mellon University 2010, *Robust group's Open Source Tutorial*, Robust group's Open Source Tutorial cover the basic steps to do acoustic training and do decoding tests of the trained acoustic model, 8th May 2010, <http://www.speech.cs.cmu.edu/sphinx/tutorial.html>.

[26] Carnegie Mellon University 2010, *Learn*, Wiki and tutorial to start to learn to use the Sphinx tools that the CMU Sphinx Team provides, 23th April 2010, <http://cmusphinx.sourceforge.net/learn/>.

[27] ORACLE, 2010, *Java SE Downloads*, Java "JRE" and Java "JDK" and platform download links, 20th April 2010, <http://java.sun.com/javase/downloads/index.jsp>.

[28] MPlayer Team 2010, *Downloads*, MPlayer download links for different architecture systems, 20th April 2010, <http://www.mplayerhq.hu/design7/dload.html>.

[29] Apache Software Foundation 2010, *Welcome*, The website of Apache Ant site, all the necessary links are the for how to download, install it and start to use it, 20th April 2010, <http://ant.apache.org/>.

[30] Carnegie Mellon University 2010, *Download*, All of the CMU Sphinx tools along with recognizers and training tools, 23th April 2010, <http://cmusphinx.sourceforge.net/download/>.

[31] Carnegie Mellon University 2010, *Statistical Language Modeling Toolkit*, The SLM toolkit is meant for large amounts of training data. If you intend to train a language model from a few dozen or even hundred sentences, please refer to the lmtool, 8th May 2010, <http://www.speech.cs.cmu.edu/SLM/toolkit.html>.

[32] Nickolay V. Shmyrev, 2010, *How to improve accuracy*, Describes ways to improve speech recognition accuracy, 9th May 2010, <http://nsh.nexiwave.com/2009/08/how-to-improve-accuracy.html>.

[33] Andy Holst, 2010, *Auto Transcript Generator*, Demonstration of the Auto Transcript Generator application, 1st August 2010, <http://www.youtube.com/watch?v=XObHlwfqagc>.

[34] Nickolay V. Shmyrev, 2010, *Testing ASR with Voxforge Database*, Nickolay demonstrates how important it is that we have open source on speech corpus that people can use to train their acoustic model and how efficient the ASR can be on the English database, 12th May 2010, <http://nsh.nexiwave.com/2010/04/testing-asr-with-voxforge-database.html>.

[35] Carnegie Mellon University Sphinx 2010, *Training an acoustic model with LDA and MLLT feature transforms*, Wiki of how you are supposed to setup SphinxTrain to enable training with LDA and MLLT feature transforms, 12 May 2010, <http://cmusphinx.sourceforge.net/wiki/ldamllt>.

# 9 Appendices

# Appendix 1 (number of pages: 3)

**Figure 1:** Google search for "+podcast +auto +generate +transcript +app +program"

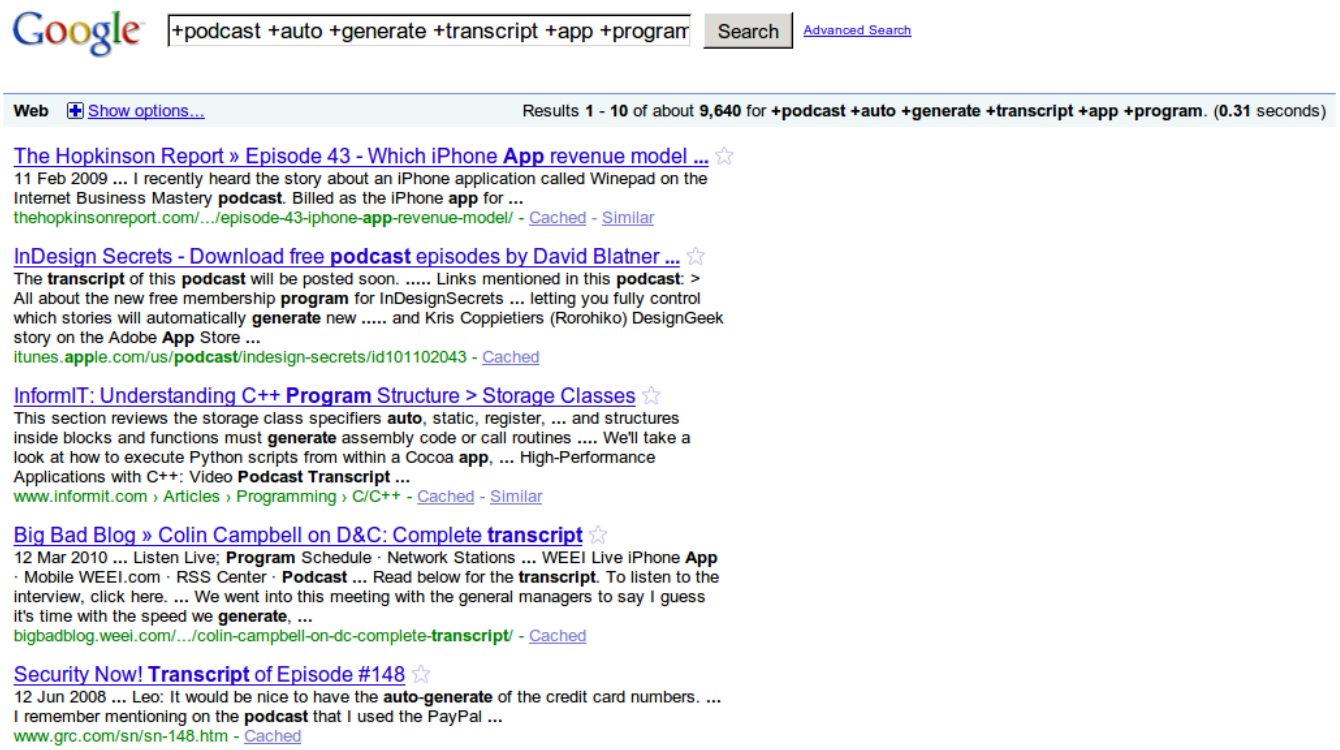


**Figure 2:** The flow chart of the Auto Transcript Generator system

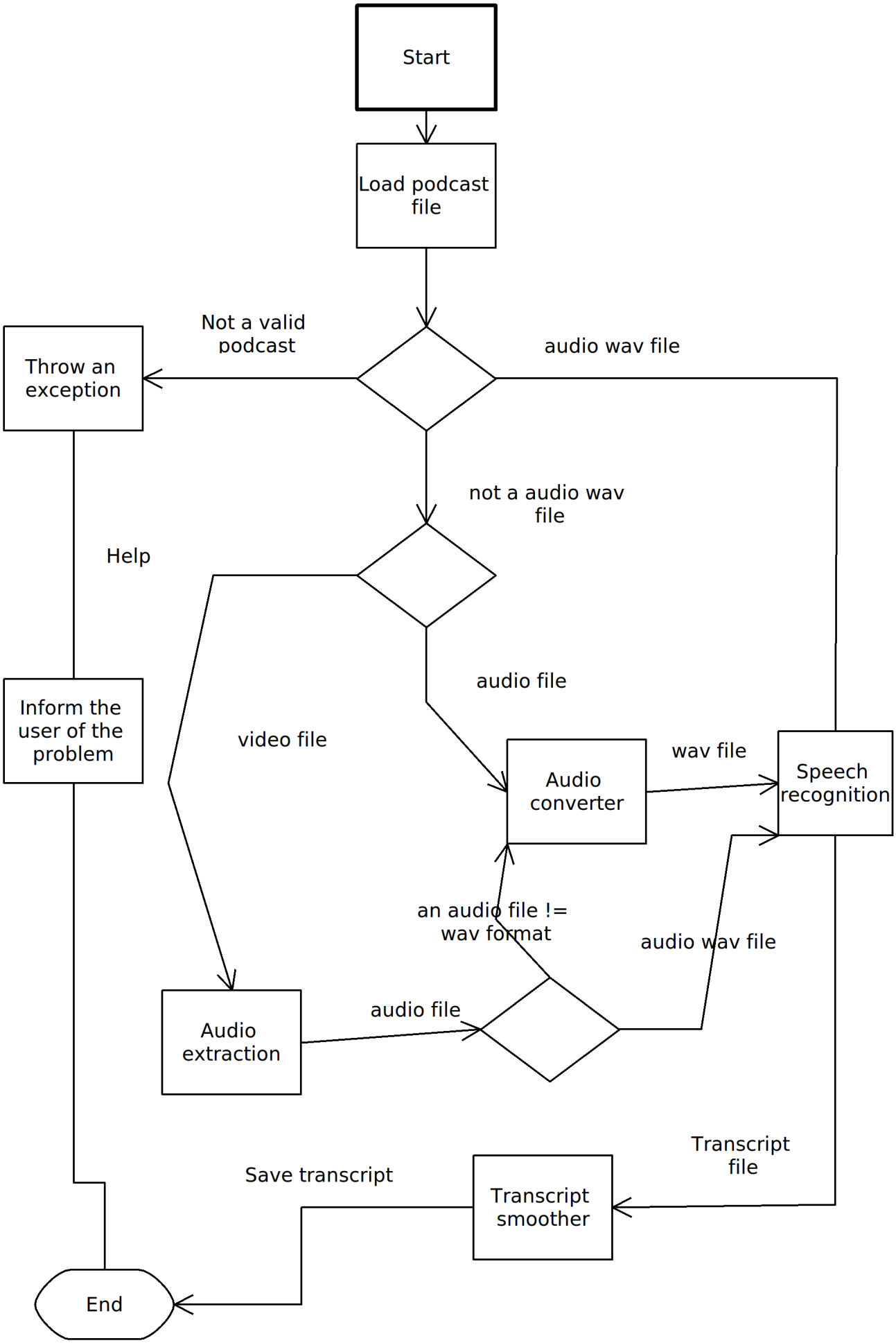
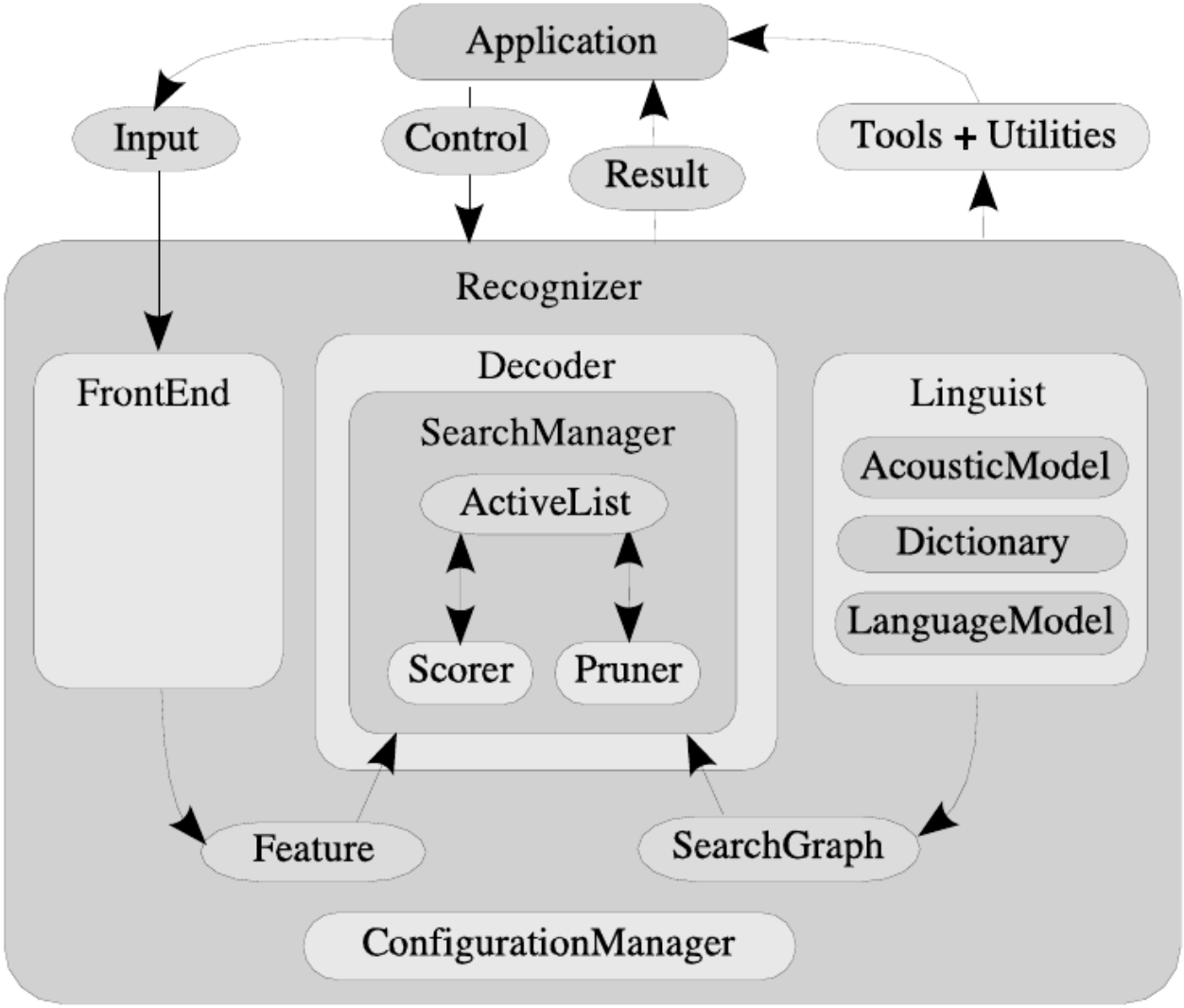


**Figure 3:** Sphinx-4 Decoder Framework. The main blocks are the FrontEnd, the Decoder and the Linguist



# Appendix 2 (number of pages: 8)

**Listing 1:** ATG's Stream Redirecter

```
class LineRedirecter extends Thread {
    /** The input stream to read from. */
    private InputStream in;
    /** The output stream to write to. */
    private OutputStream out;

    /**
     * @param in the input stream to read from.
     * @param out the output stream to write to.
     * @param prefix the prefix used to prefix the lines when outputting to the logger.
     */
    LineRedirecter(InputStream in, OutputStream out) {
        this.in = in;
        this.out = out;
    }

    public void run()
    {
        try {
            // creates the decorating reader and writer
            BufferedReader reader = new BufferedReader(new InputStreamReader(in));
            PrintStream printStream = new PrintStream(out);
            String line;

            // read line by line
            while ( (line = reader.readLine()) != null) {
                printStream.println(line);
            }
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

**Listing 2:** The basic MPlayer code setup to be able to identify tracks on podcast file.

```
    // Initiate the MPlayer process predefined settings
    this.mplayerProcess = Runtime.getRuntime().exec(start_mplayer);
    // create the piped streams where to redirect the standard output and error of MPlayer
    // specify a bigger pipesize than the default of 1024
    PipedInputStream  readFrom = new PipedInputStream(1024*1024);
    PipedOutputStream writeTo = new PipedOutputStream(readFrom);
    BufferedReader mplayerOutErr = new BufferedReader(new InputStreamReader(readFrom));

    // create the threads to redirect the standard output and error of MPlayer
    new LineRedirecter(mplayerProcess.getInputStream(), writeTo).start();
    new LineRedirecter(mplayerProcess.getErrorStream(), writeTo).start();

    // the standard input of MPlayer
    PrintStream mplayerIn = new PrintStream(mplayerProcess.getOutputStream());
```

**Listing 3:** Build.sh script for speech data preparation

```
#!/bin/sh

arg0=$1
download() {
cd tgz

wget -N -nd -c -e robots=off -A tgz,html -r -np \
 http://www.repository.voxforge1.org/downloads/SpeechCorpus/Trunk/Audio/Main/8
    kHz_16bit

cd ..
}

addSpeakers()
{
    i=0
    while read -a column
    do
  if [ $i -lt $arg0 ]; then
      $(ln -s "../../../../voxforge/audio/8k/test_extract/"${column[*]} "wav/.")
      let "i += 1"
  else
      break;
  fi
    done
}

unpack() {
for f in tgz/*.tgz; do
    tar xf $f -C wav
done
}

convert_flac() {
find -L wav -name "*flac*" -type d  | while read file; do
    outdir=${file//flac/wav}
    mkdir -p $outdir
done
find -L wav -name "*.flac"  | while read f; do
      outfile=${f//flac/wav}
      flac -s -d $f -o $outfile
done
}

collect_prompts() {
mkdir etc
> etc/allprompts
find -L wav -name PROMPTS | while read f; do
    echo $f
    cat $f >> etc/allprompts
done
#find wav -name prompts | while read f; do
#    echo $f
#    cat $f >> etc/allprompts
#done
}

#FIXME
make_prompts() {
cat etc/allprompts | sort | sed 's/mfc/wav/g' |
```

```
sed 's:../../../Audio/MFCC/XXkHz_YYbit/MFCC_0_D/::g' > allprompts.tmp
mv allprompts.tmp etc/allprompts
cat etc/allprompts | awk '{
    printf ("<s> ");
    for (i=2;i<=NF;i++)
  printf ("%s ", $i);
    printf ("</s> (%s)\n", $1);
}
' > etc/voxforge_en.transcription
./scripts/traintest.sh etc/voxforge_en.transcription
./scripts/build_fileids.py etc/voxforge_en.transcription.train > etc/voxforge_en
    .fileids.train
./scripts/build_fileids.py etc/voxforge_en.transcription.test > etc/voxforge_en.
    fileids.test
}

addSpeakers
convert_flac
collect_prompts
make_prompts
./scripts_pl/make_feats.pl -ctl etc/voxforge_en.fileids.train
./scripts_pl/make_feats.pl -ctl etc/voxforge_en.fileids.test
```

**Listing 4:** SphinxTrain Level 1

```
    # Configuration script for sphinx trainer                    -*-mode:Perl-*-

    $CFG_VERBOSE = 1;   # Determines how much goes to the screen.

    # These are filled in at configuration time
    $CFG_DB_NAME = "voxforge_en";
    $CFG_BASE_DIR = "/home/andy/TB/backup/projects/my_project/cmusphinx/trunk/
        voxforge_en";
    $CFG_SPHINXTRAIN_DIR = "../SphinxTrain";

    # Directory containing SphinxTrain binaries
    $CFG_BIN_DIR = "$CFG_BASE_DIR/bin";
    $CFG_GIF_DIR = "$CFG_BASE_DIR/gifs";
    $CFG_SCRIPT_DIR = "$CFG_BASE_DIR/scripts_pl";

    # Experiment name, will be used to name model files and log files
    $CFG_EXPTNAME = "$CFG_DB_NAME";

    # Audio waveform and feature file information
    $CFG_WAVFILES_DIR = "$CFG_BASE_DIR/wav";
    $CFG_WAVFILE_EXTENSION = 'wav';
    $CFG_WAVFILE_TYPE = 'mswav'; # one of nist, mswav, raw
    $CFG_FEATFILES_DIR = "$CFG_BASE_DIR/feat";
    $CFG_FEATFILE_EXTENSION = 'mfc';
    $CFG_VECTOR_LENGTH = 13;

    $CFG_MIN_ITERATIONS = 1;  # BW Iterate at least this many times
    $CFG_MAX_ITERATIONS = 10; # BW Don't iterate more than this, somethings
        likely wrong.

    # (none/max) Type of AGC to apply to input files
    $CFG_AGC = 'none';
    # (current/none) Type of cepstral mean subtraction/normalization
    # to apply to input files
    $CFG_CMN = 'current';
    # (yes/no) Normalize variance of input files to 1.0
```

```
35      $CFG_VARNORM = 'no';
36      # (yes/no) Use letter-to-sound rules to guess pronunciations of
37      # unknown words (English, 40-phone specific)
38      $CFG_LTSOOV = 'yes';
39      # (yes/no) Train full covariance matrices
40      $CFG_FULLVAR = 'no';
41      # (yes/no) Use diagonals only of full covariance matrices for
42      # Forward-Backward evaluation (recommended if CFG_FULLVAR is yes)
43      $CFG_DIAGFULL = 'yes';
44
45      # (yes/no) Perform vocal tract length normalization in training.  This
46      # will result in a "normalized" model which requires VTLN to be done
47      # during decoding as well.
48      $CFG_VTLN = 'no';
49      # Starting warp factor for VTLN
50      $CFG_VTLN_START = 0.80;
51      # Ending warp factor for VTLN
52      $CFG_VTLN_END = 1.40;
53      # Step size of warping factors
54      $CFG_VTLN_STEP = 0.05;
55
56      # Directory to write queue manager logs to
57      $CFG_QMGR_DIR = "$CFG_BASE_DIR/qmanager";
58      # Directory to write training logs to
59      $CFG_LOG_DIR = "$CFG_BASE_DIR/logdir";
60      # Directory for re-estimation counts
61      $CFG_BWACCUM_DIR = "$CFG_BASE_DIR/bwaccumdir";
62      # Directory to write model parameter files to
63      $CFG_MODEL_DIR = "$CFG_BASE_DIR/model_parameters";
64
65      # Directory containing transcripts and control files for
66      # speaker-adaptive training
67      $CFG_LIST_DIR = "$CFG_BASE_DIR/etc";
68
69      #*******variables used in main training of models*******
70      $CFG_DICTIONARY     = "$CFG_LIST_DIR/$CFG_DB_NAME.dic";
71      $CFG_RAWPHONEFILE   = "$CFG_LIST_DIR/$CFG_DB_NAME.phone";
72      $CFG_FILLERDICT     = "$CFG_LIST_DIR/$CFG_DB_NAME.filler";
73      $CFG_LISTOFFILES    = "$CFG_LIST_DIR/${CFG_DB_NAME}.fileids.train";
74      $CFG_TRANSCRIPTFILE = "$CFG_LIST_DIR/${CFG_DB_NAME}.transcription.train";
75      $CFG_FEATPARAMS     = "$CFG_LIST_DIR/feat.params";
76
77      #*******variables used in characterizing models*******
78
79      $CFG_HMM_TYPE = '.cont.'; # Sphinx III
80      #$CFG_HMM_TYPE  = '.semi.'; # PocketSphinx and Sphinx II
81      #$CFG_HMM_TYPE  = '.ptm.'; # PocketSphinx (larger data sets)
82
83      if (($CFG_HMM_TYPE ne ".semi.")
84    and ($CFG_HMM_TYPE ne ".ptm.")
85    and ($CFG_HMM_TYPE ne ".cont.")) {
86          die "Please choose one CFG_HMM_TYPE out of '.cont.', '.ptm.', or '.semi.',
              " .
87    "currently $CFG_HMM_TYPE\n";
88      }
89
90      # This configuration is fastest and best for most acoustic models in
91      # PocketSphinx and Sphinx-III.  See below for Sphinx-II.
92      $CFG_STATESPERHMM = 5;
93      $CFG_SKIPSTATE = 'yes';
```

```
94
95      if ($CFG_HMM_TYPE eq '.semi.') {
96        $CFG_DIRLABEL = 'semi';
97      # Four stream features for PocketSphinx
98        $CFG_FEATURE = "s2_4x";
99        $CFG_NUM_STREAMS = 4;
100       $CFG_INITIAL_NUM_DENSITIES = 256;
101       $CFG_FINAL_NUM_DENSITIES = 256;
102       die "For semi continuous models, the initial and final models have the
              same density"
103   if ($CFG_INITIAL_NUM_DENSITIES != $CFG_FINAL_NUM_DENSITIES);
104     } elsif ($CFG_HMM_TYPE eq '.ptm.') {
105       $CFG_DIRLABEL = 'ptm';
106     # Four stream features for PocketSphinx
107       $CFG_FEATURE = "s2_4x";
108       $CFG_NUM_STREAMS = 4;
109       $CFG_INITIAL_NUM_DENSITIES = 64;
110       $CFG_FINAL_NUM_DENSITIES = 64;
111       die "For phonetically tied models, the initial and final models have the
              same density"
112   if ($CFG_INITIAL_NUM_DENSITIES != $CFG_FINAL_NUM_DENSITIES);
113     } elsif ($CFG_HMM_TYPE eq '.cont.') {
114       $CFG_DIRLABEL = 'cont';
115     # Single stream features - Sphinx 3
116       $CFG_FEATURE = "1s_c_d_dd";
117       $CFG_NUM_STREAMS = 1;
118       $CFG_INITIAL_NUM_DENSITIES = 1;
119       $CFG_FINAL_NUM_DENSITIES = 4;
120       die "The initial has to be less than the final number of densities"
121   if ($CFG_INITIAL_NUM_DENSITIES > $CFG_FINAL_NUM_DENSITIES);
122     }
123
124     # (yes/no) Train multiple-gaussian context-independent models (useful
125     # for alignment, use 'no' otherwise) in the models created
126     # specifically for forced alignment
127     $CFG_FALIGN_CI_MGAU = 'no';
128     # (yes/no) Train multiple-gaussian context-independent models (useful
129     # for alignment, use 'no' otherwise)
130     $CFG_CI_MGAU = 'no';
131     # Number of tied states (senones) to create in decision-tree clustering
132     $CFG_N_TIED_STATES = 25;
133     # How many parts to run Forward-Backward estimatinon in
134     $CFG_NPART = 2;
135
136     # (yes/no) Train a single decision tree for all phones (actually one
137     # per state) (useful for grapheme-based models, use 'no' otherwise)
138     $CFG_CROSS_PHONE_TREES = 'no';
139
140     # Use force-aligned transcripts (if available) as input to training
141     $CFG_FORCEDALIGN = 'no';
142
143     # Use a specific set of models for force alignment.  If not defined,
144     # context-independent models for the current experiment will be used.
145     $CFG_FORCE_ALIGN_MDEF = "$CFG_BASE_DIR/model_architecture/$CFG_EXPTNAME.
            falign_ci.mdef";
146     if ($CFG_FALIGN_CI_MGAU eq  'yes') {
147       $CFG_FORCE_ALIGN_MODELDIR = "$CFG_MODEL_DIR/$CFG_EXPTNAME.falign_ci_${
              CFG_DIRLABEL}_$CFG_FINAL_NUM_DENSITIES";
148     }
149     else {
```

```
150       $CFG_FORCE_ALIGN_MODELDIR = "$CFG_MODEL_DIR/$CFG_EXPTNAME.
              falign_ci_$CFG_DIRLABEL";
151     }
152
153     # Use a specific dictionary and filler dictionary for force alignment.
154     # If these are not defined, a dictionary and filler dictionary will be
155     # created from $CFG_DICTIONARY and $CFG_FILLERDICT, with noise words
156     # removed from the filler dictionary and added to the dictionary (this
157     # is because the force alignment is not very good at inserting them)
158
159     # $CFG_FORCE_ALIGN_DICTIONARY = "$ST::CFG_BASE_DIR/falignout$ST::
            CFG_EXPTNAME.falign.dict";;
160     # $CFG_FORCE_ALIGN_FILLERDICT = "$ST::CFG_BASE_DIR/falignout/$ST::
            CFG_EXPTNAME.falign.fdict";;
161
162     # Use a particular beam width for force alignment.  The wider
163     # (i.e. smaller numerically) the beam, the fewer sentences will be
164     # rejected for bad alignment.
165     $CFG_FORCE_ALIGN_BEAM = 1e-60;
166
167     # Calculate an LDA/MLLT transform?
168     $CFG_LDA_MLLT = 'no';
169     # Dimensionality of LDA/MLLT output
170     $CFG_LDA_DIMENSION = 29;
171
172     # This is actually just a difference in log space (it doesn't make
173     # sense otherwise, because different feature parameters have very
174     # different likelihoods)
175     $CFG_CONVERGENCE_RATIO = 0.1;
176
177     # Queue::POSIX for multiple CPUs on a local machine
178     # Queue::PBS to use a PBS/TORQUE queue
179     $CFG_QUEUE_TYPE = "Queue::POSIX";
180
181     # Name of queue to use for PBS/TORQUE
182     $CFG_QUEUE_NAME = "workq";
183
184     # (yes/no) Build questions for decision tree clustering automatically
185     $CFG_MAKE_QUESTS = "yes";
186     # If CFG_MAKE_QUESTS is yes, questions are written to this file.
187     # If CFG_MAKE_QUESTS is no, questions are read from this file.
188     $CFG_QUESTION_SET = "${CFG_BASE_DIR}/model_architecture/${CFG_EXPTNAME}.
            tree_questions";
189     #$CFG_QUESTION_SET = "${CFG_BASE_DIR}/linguistic_questions";
190
191     $CFG_CP_OPERATION = "${CFG_BASE_DIR}/model_architecture/${CFG_EXPTNAME}.
            cpmeanvar";
192
193     # This variable has to be defined, otherwise utils.pl will not load.
194     $CFG_DONE = 1;
195
196     return 1;
```

**Listing 5:** Sphinx-3 decoder cfg file

```
1 # Configuration script for sphinx decoder                    -*-mode:Perl-*-
2
3 # Variables starting with $DEC_CFG_ refer to decoder specific
4 # arguments, those starting with $CFG_ refer to trainer arguments,
5 # some of them also used by the decoder.
6
```

```
$DEC_CFG_VERBOSE = 1;   # Determines how much goes to the screen.

# These are filled in at configuration time
$DEC_CFG_DB_NAME = 'voxforge_en';
$DEC_CFG_BASE_DIR = '/home/andy/TB/backup/projects/my_project/cmusphinx/trunk/
    voxforge_en';
$DEC_CFG_SPHINXDECODER_DIR = '../sphinx3';
$DEC_CFG_SPHINXTRAIN_CFG = "$DEC_CFG_BASE_DIR/etc/sphinx_train.cfg";

# Name of the decoding script to use (psdecode.pl or s3decode.pl, probably)
$DEC_CFG_SCRIPT = 's3decode.pl';

require $DEC_CFG_SPHINXTRAIN_CFG;

$DEC_CFG_BIN_DIR = "$DEC_CFG_BASE_DIR/bin";
$DEC_CFG_GIF_DIR = "$DEC_CFG_BASE_DIR/gifs";
$DEC_CFG_SCRIPT_DIR = "$DEC_CFG_BASE_DIR/scripts_pl";

$DEC_CFG_EXPTNAME = "$CFG_EXPTNAME";
$DEC_CFG_JOBNAME  = "$CFG_EXPTNAME"."_job";

# Models to use.
$DEC_CFG_MODEL_NAME = "$CFG_EXPTNAME.cd_${CFG_DIRLABEL}_${CFG_N_TIED_STATES}";

$DEC_CFG_FEATFILES_DIR = "$DEC_CFG_BASE_DIR/feat";
$DEC_CFG_FEATFILE_EXTENSION = '.mfc';
$DEC_CFG_VECTOR_LENGTH = $CFG_VECTOR_LENGTH;
$DEC_CFG_AGC = $CFG_AGC;
$DEC_CFG_CMN = $CFG_CMN;
$DEC_CFG_VARNORM = $CFG_VARNORM;

$DEC_CFG_QMGR_DIR = "$DEC_CFG_BASE_DIR/qmanager";
$DEC_CFG_LOG_DIR = "$DEC_CFG_BASE_DIR/logdir";
$DEC_CFG_MODEL_DIR = "$CFG_MODEL_DIR";

#*******variables used in decoding of wave files *******
$DEC_CFG_DICTIONARY     = "$DEC_CFG_BASE_DIR/etc/$DEC_CFG_DB_NAME.dic";
$DEC_CFG_FILLERDICT     = "$DEC_CFG_BASE_DIR/etc/$DEC_CFG_DB_NAME.filler";
$DEC_CFG_LISTOFFILES    = "$DEC_CFG_BASE_DIR/etc/${DEC_CFG_DB_NAME}.fileids.
    train";
$DEC_CFG_TRANSCRIPTFILE = "$DEC_CFG_BASE_DIR/etc/${DEC_CFG_DB_NAME}.
    transcription.train";
$DEC_CFG_RESULT_DIR     = "$DEC_CFG_BASE_DIR/result";

# This variables, used by the decoder, have to be user defined, and
# may affect the decoder output

$DEC_CFG_LANGUAGEMODEL_DIR = "$DEC_CFG_BASE_DIR/etc";
$DEC_CFG_LANGUAGEMODEL  = "$DEC_CFG_LANGUAGEMODEL_DIR/voxforge_en.lm.DMP";
$DEC_CFG_LANGUAGEWEIGHT = "10";
$DEC_CFG_BEAMWIDTH = "1e-80";
$DEC_CFG_WORDBEAM = "1e-40";

$DEC_CFG_ALIGN = "builtin";

#*******variables used in characterizing models*******

$DEC_CFG_HMM_TYPE = $CFG_HMM_TYPE;

if (($DEC_CFG_HMM_TYPE ne ".semi.") and ($DEC_CFG_HMM_TYPE ne ".cont.")) {
```

```
  die "Please choose one CFG_HMM_TYPE out of '.cont.' or '.semi.', " .
    "currently $DEC_CFG_HMM_TYPE\n";
}

# This comes directly from reading the code. The feature definitions
# aren're represented exactly by the same string in the trainer and
# the decoder. Therefore, we need to map between them.
%feature_type = (
     'c/1..L-1/,d/1..L-1/,c/0/d/0/dd/0/,dd/1..L-1/' => 's2_4x',
     'c/1..L-1/d/1..L-1/c/0/d/0/dd/0/dd/1..L-1/'    => 's3_1x39',
     'c/0..L-1/d/0..L-1/dd/0..L-1/'                 => '1s_c_d_dd',
     'c/0..L-1/d/0..L-1/'                           => 'cep_dcep',
     'c/0..L-1/'                                    => 'cep',
     'c/0..L-1/dd/0..L-1/'                          => 'INVALID',
     '4s_12c_24d_3p_12dd'                           => 's2_4x',
     '1s_12c_12d_3p_12dd'                           => 's3_1x39',
     's2_4x'                                  => 's2_4x',
     's3_1x39'                                      => 's3_1x39',
     '1s_c_d_dd'                                    => '1s_c_d_dd',
     '1s_c_d_ld_dd'                                 => '1s_c_d_ld_dd',
     '1s_c_d'                                       => 'cep_dcep',
     '1s_c'                                         => 'cep',
     '1s_c_dd'                                      => 'INVALID',
     '1s_d'                                         => 'INVALID',
     '1s_dd'                                        => 'INVALID',
    );

$DEC_CFG_FEATURE = "INVALID"
    unless ((exists $feature_type{$CFG_FEATURE})
      and ($DEC_CFG_FEATURE = $feature_type{$CFG_FEATURE}));

if ($DEC_CFG_FEATURE eq "INVALID") {
  die "Feature type used for training, $CFG_FEATURE, cannot be used for decoding
      .\n" .
    "Please use one of 1s_c_d_dd, 1s_c_d, 1s_c, s2_4x, s3_1x39, 1s_c_d_ld_dd\n";
}
$CFG_FEAT_WINDOW ||= 0;
# Undocumented decoder magic since SphinxBase may not support -cepwin yet
if ($CFG_FEAT_WINDOW) {
    $DEC_CFG_FEATURE = "$CFG_VECTOR_LENGTH:$CFG_FEAT_WINDOW";
}

$DEC_CFG_NPART = 2;   #  Define how many pieces to split decode in

$DEC_CFG_OKAY_COLOR = '00D000';
$DEC_CFG_WARNING_COLOR = '555500';
$DEC_CFG_ERROR_COLOR = 'DD0000';

return 1;
```