CHAPTER 2
CHAPTER 2
INPUT SIGNAL PROCESSING-SPEECH RECOGNITION SYSTEM

2.1 Introduction

This chapter deals with the input signal quality requirement for speech recognition. This chapter also, details the proposal of the research for a novel Input Signal Processing (ISP) component for Sphinx-4 to provide optimal recognition capabilities.

2.2 Speech Recognition - A Literature Survey Summary

The Intelligent process to convert the analog voice signals to the exact words uttered by the speaker may seem to be a simple task today. The research in Speech Recognition (ASR) is relatively young. Research in speech technology started at AT&T Bell Laboratories in 1936 and is going stronger [64] [68]. The first commercial automatic speech recognition system was presented in 1973 and could recognize 100 different words. The research has come a long way since then, much thanks to the computer resources and new program techniques of today.

It is only in recent years, aided by advances in processing power [40 Teraflop of Earth simulator] and memory capacity [more than 2GB], that highly capable speech recognition software has become real. The Latest announcement comes from W3C where they have framed standard for a Voice Browser. But the human auditory system is still superior to the most powerful speech recognition systems available today [126] [123].

Today, numerous research works contributed towards the development of Speech Recognition techniques and I.T Giants [Microsoft, Sun Microsystems, IBM, have taken the lead. IBM's Superhuman Speech Recognition Project aims to
develop a recognition system that meets or exceeds human performance in real-world conditions, which cannot be affected by background noise or a person's particular speech characteristics. The system, expected to be completed by the year 2010, will incorporate visual information, like lip-reading does, as well as more conventional speech-recognition techniques. Microsoft implements through its Speech API, Sun Microsystems has various implementations like Text to Speech (TTS), CMU Sphinx, Cloud garden etc. Table 2.1 shows the comparison of speech engines.

2.3 Sphinx Fundamentals

Sphinx is originated at Carnegie Mellon University (CMU) and developed by a group of faculty members and students in department of computer science in Carnegie Mellon University [125]. It is funded by DARPA (Defense Advanced Research Projects Agency) [64].

Work at CMU

A significant amount of influential work in the area of speech recognition has taken place at Carnegie Mellon University (CMU) [125]. In 1969, CMU convinced a young professor Mr. Raj Reddy at Stanford to join their computer science department. Over the next 15 years Raj Reddy would become a major figure in the worlds most successful and influential speech recognition research groups [68] [125].

Since the born the Sphinx Group is dedicated to release a set of reasonably mature, world-class speech components that provide a basic level of technology to anyone interested in creating speech-using applications without the once-prohibitive initial investment cost in research and development, the same components are open to peer review by all researchers in the field, and are used for linguistic research as well.
One of the more recent (and well known) speech recognition systems at CMU is the Sphinx system, developed in the late 80's by Lee [32] [64]. The goal of the Sphinx speech recognition system was to perform speaker independent, connected word [97], large vocabulary speech recognition. Till this time successful speech recognition systems had not achieved high performance levels at the cost of reduced task complexity, it means, they frequently simplified the problem by training and testing the recognizer on only one speaker (speaker dependent recognition), or by forcing the user to pause between words (isolated word recognition [97]).

Sphinx is the first accurate large vocabulary continuous speaker-independent speech recognition system [68] [121]. Recently, the performance of the Sphinx system was significantly improved. When dealing with increased in task perplexity, speaker variation and environment variation in speech recognition is critical for speech recognizers.

Steady progress has been made along those three dimensions [89]. Some of our recent contributions include use of additional dynamic features, speaker-normalized features, subphonetic modeling, vocabulary-independent and adaptive speech recognition, speaker adaptation, efficient search, and language modeling.

The research currently refines and extend these and related technologies to develop practical unlimited-vocabulary dictation systems, and spoken language systems for more general application domains [80] with larger vocabularies and reduced linguistic constraint. When the amount of training data is increased, the modeling error can be dramatically decreased. However, more data require different models so that more detailed acoustic-phonetic phenomena can be well characterized. Recent progress can be broadly classified into feature extraction, detailed representation through parameter sharing, application-related issues, search, and language modeling.
This work has been carried out in the context of the CMU Sphinx speech recognition system as a baseline. There are various schools of speech recognition technology [68] today, based on statistical hidden Markov modeling (HMM), and neural net technology, respectively. Sphinx uses HMM-based statistical modeling techniques [64] and is one of the premier recognizers of its kind.

2.3.1 Why CMU Sphinx

Live mode and batch mode speech recognizers, capable of recognizing discrete and continuous speech. Generalized pluggable front end architecture, includes pluggable implementations of preemphasis, Hamming window, Fast Fourier Transform (FFT), Mel frequency filter bank, discrete cosine transform, cepstral mean normalization, and feature extraction of cepstra, delta cepstra, double delta cepstra features [121].

Generalized pluggable language model architecture, includes pluggable language model support for ASCII and binary versions of unigram, bigram, trigram, Java Speech API Grammar Format (JSGF) [73], and ARPA-format FST grammars. Generalized acoustic model architecture, includes pluggable support for Sphinx acoustic models [32] [68] [121]. Generalized search management, includes pluggable support for breadth first and word pruning searches. Speech tools, includes tools for displaying waveforms and spectrograms and generating features from audio.
Table 2.1: Comparison of speech engine

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>CMU Sphinx</th>
<th>Microsoft SAPI</th>
<th>IBM Via Voice</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a. Open source</td>
<td>a. Applications and Speech Recognition engine communicate with SAPI</td>
<td>a. SMAPI is define as a part of SR which becomes the source of application</td>
</tr>
<tr>
<td></td>
<td>b. Free software, no need for initial investment</td>
<td>b. DDI and API remove implementation details making speech synthesis and SR engine and application convenient</td>
<td>b. Support 13 languages, including Cantonese and Chinese</td>
</tr>
<tr>
<td></td>
<td>c. Good for researchers and developers</td>
<td></td>
<td>c. Developers can write audio library to handle input</td>
</tr>
<tr>
<td></td>
<td>d. With relatively high quality</td>
<td></td>
<td>d. Support for Grammar Compiler APIs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>e. Support Dynamic vocabulary handling, database querying, add new words to the user’s vocabulary.</td>
</tr>
<tr>
<td>Drawbacks</td>
<td>a. Sphinx Train can be used to build any language model, it needs large volume of acoustic data and investigation of the system</td>
<td>a. Has to implement COM objects and interfaces for SR engine to be a SAPI 5 engine</td>
<td>a. Constrained input audio data format</td>
</tr>
<tr>
<td></td>
<td>b. Acoustic build process can take many days (or longer)</td>
<td>b. Limited language version</td>
<td>b. Relatively low accuracy without training</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c. Do not support grammar compiler</td>
<td></td>
</tr>
</tbody>
</table>
2.3.2 Sphinx-4 - Overview

![Figure 2.1: Existing Sphinx-4 Architecture](image)

Sphinx-4 is a state-of-the-art speech recognition system written entirely in the Java™ programming language [32] [121] [128]. It was created via a joint collaboration between the Sphinx group at Carnegie Mellon University, Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL), and Hewlett Packard (HP), with contributions from the University of California at Santa Cruz (UCSC) and the Massachusetts Institute of Technology (MIT).

Sphinx-4 evolved into a recognizer designed to be more flexible thus becoming an excellent platform for speech research [121] [128]. Figure 2.1 shows about the existing Sphinx architecture.

**The Modeling Problem**

As the complexity of tasks tackled by speech research has grown, so as that of the modeling techniques [89]. In applications, that use statistical modeling techniques, such as the Sphinx system, this translates into several tens to hundreds of megabytes of memory needed to store information regarding statistical distributions underlying the models [32].
Acoustic Models - Phones and Triphones

The objective of speech recognition is the transcription of speech into text [90]. To accomplish this, one might wish to create word models from training data. However, in the case of large vocabulary speech recognition, there are simply too many words to be trained in this way [8] [63]. It is necessary to obtain several samples of every word from several different speakers, in order to create reasonable speaker-independent models for each word. Furthermore, the process must be repeated for each new word that is added to the vocabulary. This problem is solved by creating acoustic models for sub-word units. All words are composed of basically a small set of sounds or sub-word units, such as syllables or phonemes, which can be modeled and shared across different words [53]. Phonetic models are the most frequently used sub-word models. The implementation of sub-word modeling is discussed in chapter 3. There are only about 50 phones in spoken English. New words can simply be added to the vocabulary by defining their pronunciation in terms of such phones [63]. Figure 2.2 shows the different levels of Acoustic modeling of a sample sentence.

The production of sound corresponding to a phone is influenced by neighbouring phones. IBM [31] proposed the use of triphone or context-dependent phone models to deal with such variations. With 50 phones, there can be up to $50^3$ triphones, but only a fraction of them are actually observed in practice.
Pronunciation Lexicon

The lexicon in Sphinx-4 defines [128] the linear sequence of phonemes representing the pronunciation for each word in the vocabulary. The following is a small example of the lexicon for digits:

<table>
<thead>
<tr>
<th>Digit</th>
<th>Pronunciation</th>
</tr>
</thead>
<tbody>
<tr>
<td>OH</td>
<td>s-p+iy p-iy+ch iy-ch+r sp ch-r+eh r-eh+k eh-k+ax k-ax+g ax-g+n g-n+ih n-ih+sh ih-sh+ax sh-ax+n</td>
</tr>
<tr>
<td>ZERO</td>
<td>Z IH R OW</td>
</tr>
<tr>
<td>ZERO(2)</td>
<td>Z IY R OW</td>
</tr>
<tr>
<td>ONE</td>
<td>W AH N</td>
</tr>
<tr>
<td>TWO</td>
<td>T U W</td>
</tr>
<tr>
<td>THREE</td>
<td>TH R IY</td>
</tr>
<tr>
<td>FOUR</td>
<td>F AO R</td>
</tr>
<tr>
<td>FIVE</td>
<td>F AY V</td>
</tr>
<tr>
<td>SIX</td>
<td>S IH K S</td>
</tr>
<tr>
<td>SEVEN</td>
<td>S EH V AX N</td>
</tr>
<tr>
<td>EIGHT</td>
<td>E Y TD</td>
</tr>
<tr>
<td>NINE</td>
<td>N AY N</td>
</tr>
</tbody>
</table>
There can be multiple pronunciations for a word, as shown for the word ZERO above. Each alternative pronunciation is assumed to have the same a priori language model probability.

**Word error rate (WER)**

Word error rate (WER) is a common metric of measuring the performance of a speech recognition system [127]. The general difficulty of measuring the performance lies on the fact that the recognized word sequence, which can have different length from the reference word sequence. The WER is derived from the Levenshtein distance, working at word level instead of character. This problem is solved by first aligning the recognized word sequence with the reference sequence using dynamic string alignment.

Word error rate can then be computed as,

\[
WER = \frac{S + D + I}{N}
\]

where,

- \(S\) is the number of substitutions,
- \(D\) is the number of the deletions,
- \(I\) is the number of the insertions,
- \(N\) is the number of words in the reference.

When reporting the performance of a speech recognition system, sometimes the word recognition rate (WRR) is used instead,

\[
WRR = 1 - WER = \frac{N - S - D - I}{N} = \frac{H - I}{N}
\]
Where,

- \( H \) is \( N-(S+D) \), the number of correctly recognised words.

**Equal Error Rate**

An alternative performance measure for detection tasks is the equal error rate. This is the miss (and false alarm) rate at the operating point where the two word error rates are equal [127]. Speaker identification and verification procedure is used this parameter for its implementation in chapter 6.

**2.4 The Searching Technique**

Language models range from several tens to hundreds of megabytes. There are two components to the computational cost of speech recognition are acoustic probability computation, and search. In the case of HMM-based systems [33], the former refers to the computation of the probability of a given HMM state emitting the observed speech at a given time. The latter refers to the search for the best word sequence given the complete speech input. The search cost is largely by the complexity of the acoustic models. It is much more heavily influenced by the size of the task. The search cost is significant for medium and large vocabulary recognition; it is the main focus of this research work.

If all possible sequences of words are considered, the speech recognition searching technique for the most likely sequence of words gives rise to an exponential search space. The problem has generally been tackled in two ways: Viterbi decoding [116] using beam search or stack decoding which is a variant of the A* algorithm. Some hybrid versions that combine Viterbi decoding with the A* algorithm [91].

**2.4.1 Viterbi Decoding**

Viterbi decoding [116] is a dynamic programming algorithm that searches the state space for the most likely state sequence that accounts for the input
speech. The state space is constructed by creating word HMM models from its constituent phone or triphone HMM models, and all word HMM models are searched in parallel.

Since the state space is huge for even medium vocabulary applications, the beam search heuristic is usually applied to limit the search by pruning out the less likely states. The combination is often simply referred to as Viterbi beam search. Viterbi decoding is a time-synchronous search that processes the input speech one frame at a time, updating all the states for that frame before moving on to the next frame. Most systems employ a frame input rate of 100 frames/sec [116].

Viterbi search is essentially a dynamic programming algorithm, consisting of traversing a network of HMM states and maintaining the best possible path score at each state in each frame. It is a time-synchronous search algorithm in that it processes all states completely at time t before moving on to time t+1. Figure 2.3 represents the state-time matrix network. The Viterbi algorithm is explained in 1.1.5.3.

![Figure 2.3: The state-time matrix network](image)

One dimension represents the states in the network, and the other dimension is the time axis. There is typically one start state and one or more internal states in the network. The arrows depict possible state transitions.
throughout the network. In particular, NULL transitions go vertically since they do not consume any input, and non-NULL transitions always go one time step forward. Each point in this 2-D space represents the best path probability for the corresponding state at that time. That is, given a time t and a state s, the value at (t, s) represents the probability corresponding to the best state sequence leading from the initial state at time 0 to state s at time t. The time-synchronous nature of the Viterbi search implies that the 2-D space is traversed from left to right, starting at time 0.

The search is initialized at time $t = 0$ with the path probability at the start state set to 1, and at all other states to 0. In each frame, the computation consists of evaluating all transitions between the previous frame and the current frame, and then evaluating all NULL transitions within the current frame. For non-NULL transitions, the algorithm is summarized by the following expression

$$P_j(t) = \max_i (P_i(t-1).a_{ij}.b_i(t)), \text{ } i \in \text{ set of predecessor states of } j$$

Where, $P_j(t)$ is the path probability of state j at time t, $a_{ij}$ is the static probability associated with the transition from state i to j, and $b_i(t)$ is the output probability associated with state i while consuming the input speech at t. It is straightforward to extend this formulation to include NULL transitions that do not consume any input.

Thus, every state has a single best predecessor at each time instant. With some simple bookkeeping to maintain this information, one can easily determine the best state sequence for the entire search by starting at the internal state at the end and following the best predecessor at each step all the way back to the start state. Such an example is shown by the bold arrows in figure 2.3.
The complexity of Viterbi decoding is \( N \) to \( T \), assuming each state can take transition to every state at each time step, where \( N \) is the total number of states and \( T \) is the total duration.

The application of Viterbi decoding to continuous speech recognition is straight-forward [116]. Hidden Markov Models (HMM) [87] are built by stringing together phonetic HMM models using NULL transitions between the internal state of one and the start state of the next. In addition, NULL transitions are added from the internal state of each word to the initial state of all words in the vocabulary, thus modeling continuous speech. Language model (bigram) probabilities are associated with every one of these cross-word transitions.

Note that a system with a vocabulary of \( V \) words has \( V \) to possible number of cross-word transitions. All word HMMs are searched in parallel according to

\[
P_j(t) = \max_{i} (P_i(t-1) a_{ij} b_j(t)), \ i \in \text{set of predecessor states of } j
\]

Since even a small to medium vocabulary system consists of hundreds or thousands of HMM states, the state-time matrix of Figure 2.3, quickly becomes too large and costly to compute in its entirety. To keep the computation within manageable limits, only the most likely states are evaluated in each frame, according to the beam search heuristic.

At the end of time \( t \), the state with the highest path probability \( P_{\text{max}}(t) \) is found. If any other state \( i \) has \( P_i(t) < P_{\text{max}}(t) - B \), where \( B \) is an appropriately chosen threshold or beamwidth \(< 1\), state \( i \) is excluded from consideration at time \( t + 1 \). Only the ones within the beam are considered to be active.
The beam search heuristic reduces the average cost of search by orders of magnitude in medium and large vocabulary systems. The combination of Viterbi decoding using beam search heuristic is often simply referred to as Viterbi beam search [116]. Some of the standard techniques in reducing the computational load of Viterbi search, for large vocabulary continuous speech recognition have been the following narrowing the beam width for greater pruning. However, this is usually associated with an increase in error rate because of an increase in the number of search errors: the correct word sometimes get pruned from the search path [6].

Reducing the complexity of acoustic and language models, this approach works to some extent, especially if it is followed by more detailed search in later passes. There is a tradeoff here, between the computational load of the first pass and subsequent ones. The use of detailed models in the first pass produces compact word lattices with low error rate that can be post processed efficiently, but the first pass itself is computationally expensive. Its cost can be reduced if simpler models are employed, at the cost of an increase in lattice size needed to guarantee low lattice error rates. Both the above techniques involve some tradeoff between recognition accuracy and speed.

2.4.2 Stack Decoding

Stack decoding maintains, a stack of partial hypotheses sorted in descending order of posterior likelihood. At each step it pops the best one of the stack. If it is a complete hypothesis, it is output. Otherwise the algorithm expands it by one word, trying all partial hypothesis accounts for an initial portion of the input speech. A complete hypothesis, or simply hypothesis, accounts for the entire input speech possible word extensions, evaluates the resulting (partial) hypotheses with respect to the input speech and re-inserts them in the sorted stack.
Any number of N-best hypotheses can be generated in this manner. To avoid an exponential growth in the set of possible word sequences in medium and large vocabulary systems, partial hypotheses are expanded only by a limited set of candidate words at each step. These candidates are identified by a fast match step. Since our experiments have been mostly connected [116] to Viterbi decoding, the thesis do not explore stack decoding in any greater detail.

**Tree Structured Lexicons**

Organizing the HMMs to be searched as a phonetic tree instead of the structure of independent linear HMM sequences for each word is probably the most often cited improvement in search techniques in use [87]. This structure is referred to as tree-structured lexicon [112] or lexical tree. If the pronunciations of two or more words contain the same initial phonemes, share a single sequence of HMM models representing that initial portion of their pronunciation [87]. In practice, most systems use triphones instead of just base phones, here only the triphone pronunciation sequences are considered. But the basic arguments are same. Since the word-initial models in a non-tree structured Viterbi search are typically the majority of the total number of active models, the reduction in computation is significant.

The problem with a lexical tree occurs at word boundary [112] transitions where bigram language model probabilities are usually computed and applied. In (non-tree) Viterbi algorithm there is a transition from each word ending state (within the beam) to the beginning of every word in the vocabulary. Thus, there is a fan-in at the initial state of every word, with different bigram probabilities attached to every such transition. The Viterbi algorithm chooses the best incoming transition in each case.
However, with a lexical tree structure, several words may share the same root node of the tree. There can be a conflict between the best incoming cross-word transitions for different words that share the same root node [58]. This problem has been usually solved by making copies of the lexical tree to resolve such conflicts [112].

Lexical tree structure, with a copy of the lexicon is activated for bigram transitions. All bigram transitions enter the at lexicon copy, while the backed unigram transitions enter the roots of the lexical tree. SRI notes that relying on just unigrams more than doubles the word error rate.

The recognition speed is improved by a factor of 2 to 3 for approximately the same accuracy. To gain further improvements in speed reduced the size of the bigram section by pruning the bigram language model in various ways, which adds significantly the improvement in error rate. However, it should be noted that the experimental set up is based on using HMM acoustic models, with a baseline system.

A bigram transitions constitute a significant portion of cross word transitions, which in turn are a dominant part of the search cost. Hence, the use of a lexical structure for bigram transitions must continue to incur this cost.

**Dynamic Network Decoding**

Cambridge University sphinx speech group designed a one-pass decoder that uses the lexical tree structure, with copies for cross-word transitions, but instantiates new copies at every transition, as necessary. Basically, the traditional re-entrant lexical structure is replaced with a non-re-entrant structure. To prevent an explosion in memory space requirements, they reclaim HMM nodes as soon as they become inactive by falling outside the pruning beamwidth.
Furthermore, the end points of multiple instances of the same word can be merged under the proper conditions, allowing just one instance of the lexical tree to be propagated from the merged word ends, instead of separately and multiply from each.

The number of active HMM models per frame in the scheme is actually higher than the number in the baseline Sphinx system under similar test conditions, except that Sphinx uses a different lexicon and acoustic models [45]. There are other factors at work, but the dynamic instantiation of lexical trees certainly plays a part in this increase. The overhead for dynamically constructing the HMM network is reported to be less than 20% of the total computational load. This is actually fairly high since the time to decode a sentence on an HP735 platform is reported to be about 15 minutes on average.

Even with the beam search heuristic, straight forward Viterbi decoding is expensive. The network of states to be searched is formed by a linear sequence of HMM models for each word in the vocabulary. Lexical trees can be used to reduce the size of the search space. Since many words share common pronunciation prefixes, they can also share models and avoid duplication. Recently, Sphinx-4 team has been introduced in the main search component of several systems.

The main problem faced by the researchers is in using a language model. Normally, transitions between words are accompanied by a prior language model probability. But with trees, the destination nodes of such transitions are not individual words but entire groups of them, related phonetically but quite unrelated grammatically. An efficient solution to this problem is one of the important contributions of the proposed research work.

2.4.3 Multipass Search Techniques

Viterbi search algorithms usually create a word lattice in addition to the best recognition hypothesis. The lattice includes several alternative words that
were recognized at any given time during the search. It also typically contains other information such as the time segmentations for these words, and their posterior acoustic scorers that is, the probability of observing a word, given the time segment of input speech. The lattice error rate measures the number of correct words missing from the lattice around the expected time. It is typically much lower than the word error rate of the single best hypotheses produced for each sentence.

Word lattices can be kept very compact, with low lattice error rate, if they are produced using sufficiently detailed acoustic models as opposed to primitive models word error rates are measured by counting the number of word substitutions, deletions, and insertions in the hypothesis, compared to the correct reference sentence. This is important for the practical applicability of techniques.

Lattices can be created with low computational overhead if research use simple models, but their size must be large to guarantee a sufficiently low lattice error rate. On the other hand, compact, low-error lattices can be created using more sophisticated models, at the expense of more computation time. The efficient creation of compact, low-error lattices for efficient post processing is another byproduct of this work.

2.5 Proposed system to enhance the Audio/Utterance detection

The research propose an enhancement in Speech Recognition implementation followed by CMU Sphinx to include an Input signal Processing which generates clear Acoustic Signal format which forms the basis for elegant and efficient Speech Recognition. Sphinx-4 follows a modular architecture with a front end which will collect the audio signals and pass it on to the application which has an underlying decoder which will refer the Knowledge base to transcribe [86] the spoken word.
Input Signal Processing (ISP)

The component proposed by us deals with transforming the input signal received from the Front-end into the better utterance format, the acoustic scorer can process. The ISP regenerate the Input signal using a mechanism of adding more clarity and filter unambiguous signal to identify the best format and the speech recognizer can understand and process the signal to the subsequent modules.

This is a pioneer work in the category of Input signal processing as there is no calibration scale or standard available for reference as on date to determine the speech rate requirement for better speech recognition. The proposed research work is carried out to define that speech rate plays an important role in better speech recognition and an optimal speech rate will consistently give an effective speech recognition results.

ISP Mechanism

The proposed research used Sphinx-4 a HMM based Speech Engine for continuous speech recognition for easy conversational transformation. Another open source application is Edinburgh University's Festival and general speech synthesis system. Sphinx model is a very flexible and it stood as an example for the research purpose, configured the Sphinx-4 to be very optimal and also extended a new module ISP (Input signal processing), a module which works as a preprocessor to the speech signal for maximum conversational transformation. Figure 2.4 shows the proposed ISP with Sphinx-4 Architecture.

The three aspects of performance, recognition speed, memory resource requirements, and recognition accuracy, are in mutual conflict. It is relatively easy to improve recognition speed and reduce memory while trading away some accuracy, for example by pruning the search space more drastically, and by using simpler acoustic and language models. Alternatively, one can reduce memory
requirements through efficient encoding schemes at the expense of computation time needed to decode such representations, and vice versa. But it is much harder to improve both the recognition speed and reduce main memory requirements while preserving or improving recognition accuracy. The research work demonstrated algorithmic and heuristic techniques to tackle the problem.

![Figure 2.4: Proposed ISP with Sphinx-4 Architecture](image)

The Input signal processing component proposed by us deals with transforming the input signal received from the Front-end (Input signal acceptor module) into the better utterance format (Decipherable format) by using an efficient algorithm based on speech rate which makes an intelligent rationalization of the speed of the Input signal over accuracy and suggests an optimal speed for better acoustic scorer performance (Logical processor in Speech recognition) thereby improving the overall WER and generates a robust system with more precision towards speech.

When the recognizer starts up, it constructs the front end (which generates features from speech [129]), the decoder, and the linguist (which generates the search graph) according to the defined configuration. These components will in turn construct their own subcomponents. The linguist will construct the acoustic
model, the dictionary, and the language model. It will use the knowledge from these three components, to construct a search graph that is appropriate for the task. The decoder will construct the search manager, which in turn constructs the scorer, the pruner, and the active list. The research uses N-Gram grammars which has the advantage of being able to cover a much larger language than would normally be derived directly from a corpus. Open vocabulary applications are easily supported with N-Gram grammars. Depth-first search is similar to conventional stack decoding. The most promising tokens are expanded in time sequentially.

Thus, paths from the root of the token tree to currently active tokens can be of varying lengths. In breadth-first search, on the other hand, all active tokens are expanded synchronously, making the paths from the root of the tree to the currently active tokens equally long. The proposed research work has chosen the Breadth first search method as it suits for research well, which was verified through several training experiments.

One example of a property is the sample rate of the incoming speech data. The active list is a component that requires explanation. There can be many possible paths through the search graph. The proposed research has implemented a token-passing algorithm for finding the paths through the search graph. Each time the search arrives at the next state in the graph, a token is created. A token point will point to the previous token, as well as to the next state or token. The active list keeps track of all the current active paths through the search graph by storing the last token of each path. A token has the score of the path at that particular point in the search. To perform pruning, the research work simply prune the tokens in the active list.

When the application asks the recognizer to perform recognition, the search manager will ask the scorer to score each token in the active list against the next feature vector obtained from the front end. This gives a new score for each of the
active paths. The pruner will then prune the tokens (that is, active paths) using certain heuristics. Each surviving paths will then be expanded to the next states, where a new token will be created for each next state. The process repeats itself until no more feature vectors can be obtained from the front end for scoring. This is usually means that there is no more input speech data. At that point, the works look at all paths that have reached the final exit state, and return the highest scoring path as the result to the application. Table 2.2 Shows words and vocabulary system. The proposed research used TI connected digits to test in a regressively way for the application recognition. The Sphinx has provided the following models [62]. The research has chosen connected digits as it has a scope for continuous speech recording and recognition in a real time mode.

**Table 2.2: Various Vocabulary system and Digits**

<table>
<thead>
<tr>
<th>Vocabulary system</th>
<th>Digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolated Digits</td>
<td>TI46</td>
</tr>
<tr>
<td>Connected Digits</td>
<td>TIGITS</td>
</tr>
<tr>
<td>Small Vocabulary</td>
<td>AN4</td>
</tr>
<tr>
<td>Medium Vocabulary</td>
<td>RM1</td>
</tr>
<tr>
<td>Large Vocabulary</td>
<td>HUB4</td>
</tr>
</tbody>
</table>

Lippmann[69] has provided an interesting comparison between human and machine performance in terms of word error rate for different tasks, from TI connected digits to phrases from Switchboard telephone conversations, all in the talker-independent mode.

Lippmann[69] concludes that even the presently best single systems for specific tasks, varying from 10-word to 65,000-word vocabularies, are still one or more orders of magnitude worse than human performance on similar tasks. Lippmann [69] suggests that the human-machine performance gap can be reduced
by basic research on improving low-level acoustic-phonetic modeling, on improving robustness with noise and channel variability [59], and on more accurately modeling spontaneous speech. The Research work is based on improving the audio signal quality requirement for efficient speech recognition working on the speech rate.

The ISP model is proposed with the confidence that the input signal speed can be fine tuned for better realization by the Speech Engine (Sphinx). To explain it technically the research configured the Result.getFrameNumber() function in the Result class by multiplying with the windowShiftInMs (a property of edu.cmu.sphinx.frontend.window.RaisedCosineWindower), which 10 milliseconds by default, to get the length of the result.

Since the research proves that the largest improvement on the recognition of fast speech is in the better match with the SI phone duration and dynamic features trained from regular speaking rate speech, the research use a smaller window shift in generating the cepstrum, the speed factor ρ is first computed as formula outlines with the default window shift (in our case 10ms). Then a new window shift is computed as inversely proportional to ρ,

\[ s' = \frac{s}{\rho} \]

The number of resulting windows depends on the window_size and the window_shift (commonly known as frame shift in speech world). The thesis also has taken a standard reference, to identify the silence as well as the speech based on several repeated experiments. The research work has also lowered the speech classifier 'threshold' property in the configuration files to make the input signal to be loud enough for the Sphinx engine to recognize. By using repeated experiment results, the research have set a standard for optimal speed for the input signal and configured the module (ISP) which identifies whether the speaker is speaking
slowly or quickly and do the necessary performance modification to increase the efficiency.

2.6 Comparison of Results

The research work has implemented in a Railway helpdesk application based on the HMM, Sphinx-4 and incorporated the ISP model. The application was developed with the scope of helping the railway help line for customer frequent queries about the Train Number, Platform number, Departure time, Route, frequency in a week etc., Figure 2.5a. & 2.5b shows the Screen generated by the application incorporated with ISP.

Figure 2.5a: Screen generated by the application incorporated with ISP
Experimental Setup

The following clearly depicts the experimental setup.

The application was developed for Railway Help desk using Sphinx-4 (Pure Java).
The application will recognise numbers and words like train number, platform, station, train name, destination and generates result. The system setup is

- Operating system - Windows 2000 Server
- Application tool - JDK 1.4.2/Sphinx-4/ANT/SQLServer
- Processor – Pentium IV with 256MHZ
- Headphone with Mic – Frontech
- Search Manager- SimpleBreadthFirstSearchManager
- Linguist - flat Linguist
- Scorer - threaded scorer
- Front end - pipeline

With ISP mechanism the research will trace the speech rate by identifying the time taken for the utterance and compare with the optimal speech rate (30-35ms) which is taken as the standard speech rate after repeated experiments with the model. If the value exceeds the standard, the research invoke appropriate methods to acquire the best results out of the utterance.

The research use models trained from SphinxTrain for Indian standards, packaged them into a JAR file. The advantage of having it in a JAR file is that the JAR file can simply be included in the class path and referenced in the configuration file for it to be used in a Sphinx-4 application. Further speaker independent training is not required. Table 2.3 and Figure 2.6 shows the results of speech recognition with input signal processing and without input signal processing.
Table 2.3: Results comparison of Sphinx-4 with ISP and without ISP

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Description</th>
<th>Vocabulary. Size</th>
<th>Recogn. perplex</th>
<th>% word error With ISP</th>
<th>% word error Without ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI digits</td>
<td>read digits</td>
<td>15</td>
<td>15</td>
<td>0.56</td>
<td>0.8588</td>
</tr>
<tr>
<td>Alphabet</td>
<td>read letters</td>
<td>28</td>
<td>28</td>
<td>4.2</td>
<td>5.77</td>
</tr>
</tbody>
</table>

Figure 2.6 Results comparison of Sphinx with ISP and without ISP

Based on our experiment, speaking rates are defined by the length of an individual phone relative to its average duration in the SI-284 training corpus. The research has used SI-284 corpus as it best suits our research of analyzing speech rate for speaker independent scope that a simple measure of the number of phones per second is not informative enough, and that some knowledge or estimation of the phones that were uttered will improve the estimation of real speaking rates.
The Figure 2.7 shows, the phone duration, a Gamma distribution assumption is a closer fit to the real distribution than is a Gaussian distribution. Note also that the minimum duration for any phone is 30 ms which is dependant on the HMM topology [87] was used. In order to estimate the speaking rate of a testing utterance, a first pass recognition is run on the utterance, and phone segmentation information is recorded. By comparing the phone duration with those from the SI-284 (corpus) statistics, the thesis can stretch the testing utterance either phone-by-phone or sentence by sentence.

**Phone-by-phone Length Stretching**

To stretch on a phone-by-phone basis, each phone segment is adjusted in length to the peak of the Gamma distribution of the phone in the SI training corpus:

\[
\Gamma(x, \alpha, \beta) = \frac{\beta^\alpha x^{\alpha - 1} e^{-x\beta}}{\Gamma(\alpha)}
\]

Let \( \Gamma(x, \alpha_i, \beta_i) \) be the distribution for phone i, then since its mean is \( \mu_i = \alpha_i / \beta_i \) and the variance is \( \alpha_i / \beta_i^2 \), the research work use the training data to approximate the true mean and variance in order to estimate parameter \( \alpha_i \) and \( \beta_i \). The peak of the
Gamma distribution occurs at peak, and the research work define the length-stretching factor (\( \rho_i \)) for a phone segment with length \( l_i \) as:

\[
peak_i = \frac{\alpha_i}{\beta_i} \quad \rho_i = \frac{peak_i}{l_i} \quad \ldots \ldots \quad (2.1)
\]

Notice for fast speech, \( \rho_i > 1 \), and for slow speech \( \rho_i < 1 \). Our first attempt showed that this phone-by-phone duration adjustment did not yield any recognition improvement if the correct phone sequence is unknown. This is due to the fact that a wrong phone identification and segmentation might adjust that segment of speech in the wrong direction. Therefore, although great improvement was achieved when the correct phone sequence is known, the thesis decided not to use this approach.

**Sentence-by-Sentence Stretching**

To stretch on a sentence-by-sentence basis, there are many methods to compute a single normalization factor \( \rho \) to apply to the entire utterance. One method is to find \( \rho \) which maximizes the joint probability of the utterance with respect to the SI phone duration probability distributions. That is,

\[
\bar{\rho} = \arg \max_{\rho} \left\{ P(\rho l_1 | r_1) \cdot P(\rho l_2 | r_2) \cdots P(\rho l_n | r_n) \right\}
\]

or alternatively

\[
\rho = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} \beta_i l_i}
\]

where, \( n \) is the number of phone segments in an utterance. Again preliminary experiments showed this approach failed to make improvements, given that HMM's treat each frame of speech equally important, perhaps the
influence of a phone segment should be proportional to its duration. Because of
the above reasons, the final technique adopted as AveragePeak, for determining
the best sentence based normalization factor $\rho$ is to simply average all the phone
by-phone peak factors:

$$\rho = \frac{1}{n} \sum_{i=1}^{n} \rho_i$$

(2.2)

where, $\rho_i$ is defined by Formula (2.1). This smoothing/averaging effect
compensates for the mistakes in the phone sequence estimation and indeed
provides us a stable improvement on fast speech and no degradation on regular-
speed speech.

In this experiment, the proposed system applied the concept of ISP only
to the train data. The algorithm first estimated the phone segments in the testing
utterance by running the decoder. Then used the hypothesized phone segments to
find the sentence-based normalization factor $\tilde{\rho}$. Table 2.4 and Figure 2.8 shows
16.64% error rate reduction by interpolating the cepstrum frames on dev-fast train
data. The normalization factor $\rho$ was determined by Average Peak as defined by
formula (2.2). The normalization factors of the utterances in dev-fast (set of out of
vocabulary words) varied between 0.92 and 1.47.

**Table 2.4: Word error rates on dev-fast with MFCC interpolation.**

<table>
<thead>
<tr>
<th>Training data</th>
<th>Original</th>
<th>Interpolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>16.64%</td>
<td>13.90%</td>
</tr>
</tbody>
</table>
2.7 Summary

This chapter has given an enhancement to the existing Sphinx-4 architecture to include an Input signal processing for generating the quality input signal for Speech Recognition. Highlighted the issues of Quality Input signal Requirement for efficient Speech Recognition. Also provided the recent developments by all Developers giving thrust to the same through various phases including the hardware. The approach has a significant advantage as it leverage the existing Sphinx-4 architecture. The research justifies that as economical and efficient solution.