Dependability of CMUSphinx Speech Recongition

Charles Ferraro
University of Virginia
cbf6yd@virginia.edu

***Abstract*—Speech recognition is being increasingly utilized in everyday use due to the large amount of utility it presents for accessibility and convenience. This paper will focus on speech recognition engine called Pocketsphinx that is part of the CMUSphinx toolkit. An insight on the models used along with the effects of noise fault injection into audio files will be explored as well as potential avenues for improvement.**

Keywords—speech recognition, phones, n-gram language model, relative frequency, fault injection

I. Introduction

Currently audio dictation is becoming increasingly popular as the versatility of the technology continues to improve. The technology presents many benefits from automatically creating accessible captions for hearing impaired users along with providing a helpful visual tool for assistance in comprehension. However, the technology is still prone to improper dictation and false positives. Often these errors are caused by other vocal interference, machine or digital noise, and the accent or unique speech pattern some individuals possess.

This paper focuses on the interference generated by machine or digital noise. To investigate the reliability of voice recognition/dictation the Carnegie Mellon University (CMU) Sphinx speech recognition platform will be used. The CMUSphinx toolkit is an old yet reliable speech recognition toolkit used to create speech dictation applications. The version of CMUSphinx that will be used is Pocketsphinx [1].

Pocketsphinx is a lightweight speech recognition engine that is made primarily for handheld mobile devices or low-performance desktop computers. To function it does not require an internet connection nor significant processing power. This allows the entire speech recognition engine self-contained [1]. This engine was selected particularly for the modularity it provided as well as a large amount of user control it has.

Another tool used in this paper is, is the knowledge set used to generate the n-gram probabilities and pronunciation dictionaries. This tool, also made by Carnegie Mellon University, is called the Sphinx Knowledge Base Tool. It is based on the English language with a focus on American dialect [2].

II. Related Works

Fault injection has been a common tool to improve and test the reliability of speech recognition systems. Often faults can range from utterances that are phonetically vague as well as noise that are commonly found in human environments. Presented in Veisi et al. is an analysis and experimentation of fault tolerant techniques in speech recognition systems with the intention to make these systems robust to noise. Duplication-with-comparison and NMR methods are experimented with to determine if these methods result in an improved fault-tolerant design in speech recognition systems [3].

Another paper regarding faults in speech recognition system is the research paper created by Vargas et al. which presents a speech recognition system built to reconstruct noisy signals. This fault-tolerance is achieved by implementing a concurrent consistency check (CCC) for every multiply-and-accumulate (MAC) operations. What this does is avoid an overflow of successive MAC operations when the speech recognition system is exposed to noise. In addition, a transparent Built-In Self Test (BIST) is implemented inside the pattern matching & logic decision block [4].

III. System Overview

Pocketsphinx is composed of a series of models that are used to form the bulk of its speech recognition abilities. These models collectively form a pipeline that transforms the data at each stage and passes it to the next model. In the beginning of the pipeline, it is composed of an acoustic model that is capable of effectively breaking down continuous human speech into individual pieces of data. This division of data is thereafter passed onto the next model. Which is a language model that is utilizes a phonetic dictionary to determine the likeness of which word is being uttered by the user given the first word said.

IV. Acoustic And Phonetic Model

Before an n-gram statistical language model is used to determine the likelihood of which words were uttered by the user, the audio must be processed and decomposed into processable data. Speech is a continuous stream of words composed of stable states and dynamically changing states. From these states human speech can be decomposed into a series of recognizable sounds referred to as phones [5]. These phones are only approximations of sounds said by humans, and are subject to error based on homophones, accents, and noise. Phones explicitly include a combination of consonants or vowels found in human speech which compose words. Another categorization of speech that remains distinct from phones is the principle of fillers. While phones have meaningful data relating to the words said, fillers do not. Fillers are often utterances without meaning which can include a breath or a cough [5].

Pocketsphinx, and the greater sphinx toolset, recognizes the words spoken by splitting speech into separate utterances which are divided by the short pauses in between words. To optimize the amount of processed audio, the speech of a user is divided into frames which are typically 10 milliseconds in

length of the original audio. From each frame 39 numbers that represent speech will synthesized to form a feature vector [5]. A feature vector is a n-dimensional vector of numerical features that represent an object [6]. These vectors are ultimately used with various models that compose a speech recognition system to perform dictation.

A phonetic dictionary is composed of phones that is generated from words inside of a corpus. A corpus is collection of human sentences or phrases over a selection of human textual information. This can include anything from books, plays, or transcribed conversations. From the words in corpus a series of phones is generated that map to the respective word. An example of the phonetic dictionary that used in this project is shown in Table 1 below. This phonetic dictionary was generated from Sphinx Knowledge Base Tool [7]. In addition, this section of the phonetic dictionary was used during the simulations of one of the relevant voice lines.

Table 1

Corpus Words	Phonetic Equivalent
BOWLS	B OW L Z
IN	IH N
IS	IH Z
OFTEN	AO F T AH N
OFTEN(2)	AO F AH N

It should be noted that in Table 1 differing ways to speak the word "often" are included in the generated phonetic dictionary to account for pronunciation differences from human speech. It is based on this phonetic division that the language model matches up the phones uttered with textual word that is used to represent those phones.

V. N-GRAM LANGUAGE MODELS

Because of Pocketsphinx's commitment to modularity and performance it uses an N-gram language model. Language models like these works by generating a probability of the next possible world spoken by the user based on the previous words the user said. As an example, if a user says the following half of a sentence, "Please may you get the-" it is far more likely that it will end with the word "phone" than the world "please." It is this fundamental principle that the language model is built on [8].

The probability for each successive word given the previous words spoken by the user can be performed by utilizing the chain rule of probability which states the following [8].

$$P(X_1 \dots X_n) = P(X_1)P(X_2|X_1)P(X_3|X_{1:2}) \dots P(X_n|X_{1:n-1})$$

$$= \prod_{k=1}^{n} P(X_k|X_{1:k-1}) \quad (1)$$

Now it is possible to exchange out the variables in Equation 1 with the variables used to represent words and past words. These variables are shown in Equation 2 and Equation 3 respectively.

$$P(w_n) \quad (2)$$

$$P(w_{1:n-1}) \quad (3)$$

Substituting these values gives us the equation that can be used to calculate a probability of the next word being said by the user. This new equation is shown in below in Equation 4.

$$= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \quad (4)$$

This equation is used as its superior to the alternative approach to generating probabilities of n-grams of words. The ordinary approach is to count up all the occurrences and combinations of certain n-grams existing within a knowledge set, which takes a lot of computation and time to generate. Instead, Equation 4 is combined with another equation that is used to generate a relative frequency. A relative frequency is calculated by dividing the frequency of a sequence with the observed frequency of the prefix in that sequence [8]. For example, the sentence "Rice is often served in round bowls" will have each n-gram, usually bigrams consisting of two words, divided by the frequency the word "Rice" shows up in a large established corpus/knowledge set. This relative frequency is shown in Equation 5 below.

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}w_n)}{C(w_{n-N+1:n-1})} \quad (5)$$

A table showcasing the generated probabilities of all the bigrams contained in the three sentences "Rice is often served in round bowls," "Rice is often served with chopsticks," and "Rice is sometimes white" is shown below in Table 1. These selected bigrams probabilities are an excerpt from the generated n-gram probabilities from the Sphinx Knowledge Base. Bigrams that are guaranteed to be said given the prefix is said have a higher weighted probability while bigrams that have shared prefixes which have a lower weighted probability. Highlighted in Table 2 are two bigrams that share the same prefix, which results in differing probability weight for both bigrams.

Table 2

Bigram Words	Prob. of Relative Occurrence
<s> RICE	-0.3010
BOWLS </s>	-0.3010
IN ROUND	-0.3010
IS OFTEN	-0.4771
IS SOMETIMES	-0.7782
OFTEN SERVED	-0.3010
RICE IS	-0.3010
ROUND BOWLS	-0.3010

As seen in Table 2 the bigram "is sometimes" also has a lower probability weight than the bigram "is often." As mentioned before, this is due to the phrase "is often" occurring more times in the Sphinx Knowledge Base than "is sometimes." Meaning among the literature references that

compose the Sphinx Knowledge Base one phrase occurs more frequently than the other resulting in a different weighted probability. Using these probabilities and piping the data generated from the acoustic model allows for speech recognition to finally be performed in Pocketsphinx.

VI. Approach

In order to inject faults into Pocketsphinx and be able to observe the effects of these faults, direct modification of the audio used during speech recognition needed to be performed. By modifying the audio files used in dictation we would be able to compare the effects of the modification with the unaltered audio file. From these modifications we can thereafter determine how and in what way Pocketsphinx's speech recognition breaks. These series of experiments were performed by using a Linux bash script in which the fault injection was performed automatically on the relevant audio files of a person speaking. Shown in Figure 1 is a high-level overview of this Linux bash script.

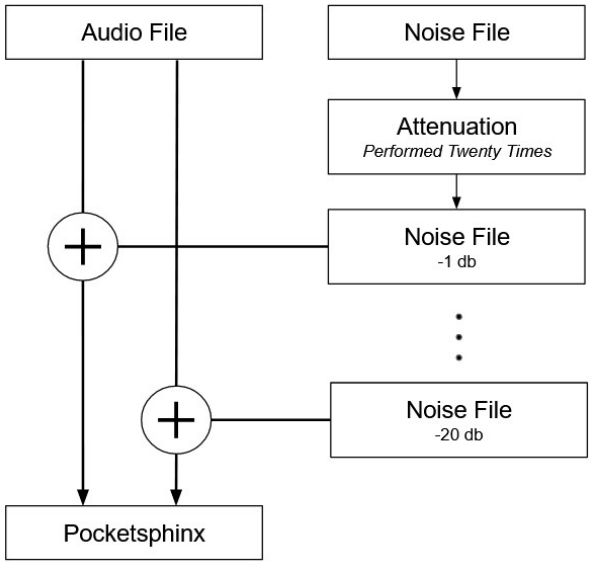

Fig. 1. Outline of Linux Experimental Script. The attenuation process occurs twenty times creating twenty separate noise files that are each seperately summed with the original audio file containing speech.

First, the Linux bash script begins by attenuating the gain of one of the three noise files by -1 dB successively twenty times each. These twenty noise files are then separately summed with the original audio file. Once all of these files are combined with the original audio file, they are piped directly into Pocketsphinx. Thereafter, the speech recognition software will attempt to transcribe the speech into words despite the noise. Depending on how many words it gets wrong or omits will determine the reliability of Pocketsphinx given that amplitude of noise. It is anticipated that low amplitude noise will result in less faults generated by the speech recognition software than high amplitude noise.

This Linux bash script takes 3 separate sentences said by a speaker and automatically applies three separate types of audio noise to the file. These sentences were the following: "Rice is often served in round bowls", "The birch canoe slid on the smooth planks", and "Glue the sheep to the dark background." Each of these sentences has three alternations of the same sentence included in the corpus meant to establish a series of possible bigrams that share the same prefix. Without the inclusion of other alternative sentences, the probability weights generated from Sphinx Knowledge Base would all be equal rendering the n-gram language model useless.

VII. Experimental Results

The simulation was executed and all three noise files at gradually increasing amplitudes were summed with the original audio files. Shown in the three graphs are the effects of each noise file onto the ability for Pocketsphinx to successfully trancribe each file accurately. It should be noted that all noise files roughly have the same amplitude. The only noise file that has a slightly inconsistent amplitude is the white noise file with a gaussian distrubtion.

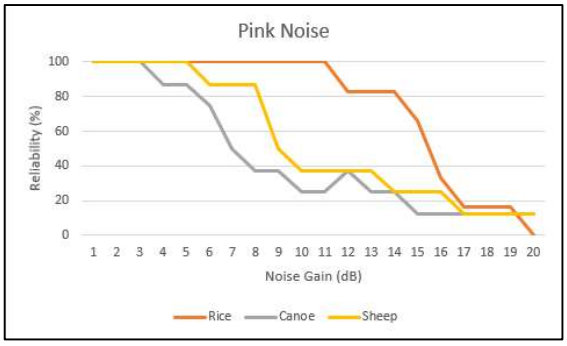

Fig. 2. Pink Noise Affect on Pocketsphinx Accuracy. Here twenty files with varying amplitudes of pnik noise was summed with the original audio file of the three sentences used in this experiment.

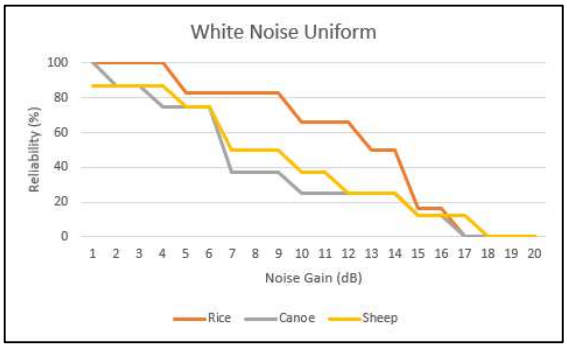

Fig. 3. White Noise with Uniform Distrubtion Affect on Pocketsphinx Accuracy. Here twenty files with varying amplitudes of pnik noise was summed with the original audio file of the three sentences used in this experiment.

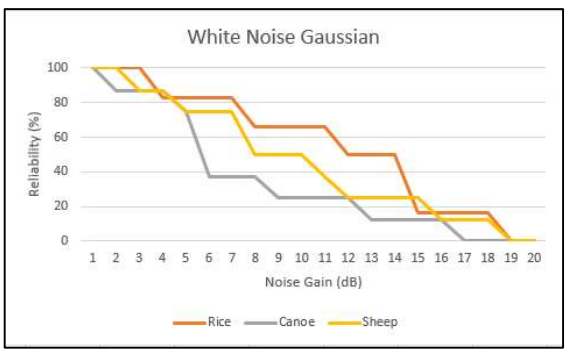

Fig. 4. White Noise with Gaussian Distrubtion Affect on Pocketsphinx Accuracy. Here twenty files with varying amplitudes of pnik noise was summed with the original audio file of the three sentences used in this experiment.

An interesting pattern that persisted through the experiment was that the words most suspectible to faults were the bigrams that shared a prefix. As evident with the canoe sample setence and the sheep sample sentence there are consistent dips among all audio files that occur at 6-9 dB of noise gain. These sharp dips at this interval were due to these bigrams being the first to be incorrectly trancribed. This is likely due to the principle evident in Table 2 wherein these bigrams experienced different probability weights than the rest of the other words. This means that with greater inteference, Pocketsphinx ability to successfully choose one bigram over another was weaker than compared to bigrams that did not share prefixes.

After these bigrams are incorrectly guessed the number of incorrect guesses for each increasing amount of noise gain became more gradual as single words began to be omitted or improperly transcribed. This phenomenon appears to have occurred for the rice sample sentence albeit much later in the simulation. A possible explanation for this might be that the phonetic pronunciation in the sound file may be much more distinct or clearer than the other two sample sentences. This may have given the n-gram language model a much better basis to guess which word will be said next.

VIII. Modifications

A possible modification that can be used to increase the reliability of Pocketsphinx may be creating a self-correcting algorithm that factors in the probability of higher n-grams over bigrams. Presented in this research paper created by Hrinchuk et al. is an example of a correction algorithm that uses a 6-gram language model for post-processing correction. This would allow Pocketsphinx to analyze the words spoken after a prefix to determine what was previously said. While this would slightly compromise the speed and strict adherence to real-time processing in Pocketsphinx, it would nonetheless improve transcription accuracy [9]. This increase in transcription accuracy would be invaluable towards properly determining speech commands with different arguments or what the user is attempting to transcribe in a document wherein the corpus is very large.

Additionally, another modification that can be used to improve Pocketsphinx resistant to noise is another paper presented by Hermus et al. in which the noise present in an audio file is decomposed with the assumption that human speech is based on a low-rank linear model while the uncorrelated additive noise is based on white noise interference. This makes the assumption that the noise present follows are a particularly well-defined model [10]. Having used white noise to create faults in Pocketsphinx this methodology could be very valuable in order to resist persistent and consistent noise. However, a noted weakness present in this research paper is the elimination of random or sudden noise which does not follow a well-defined model.

IX. Conclusions

While Pocketsphinx is an old speech recognition software it nonetheless has versatile use as a simple offline speech recognition software. The performance benefits enable it to be used in embedded systems or low performance computers. Given the faults created it does exhibit some level of weakness to persistent noises, and upon the infliction of this noise bigrams that share a prefix are especially weak to mistranslations. Possible modifications include using higher n-grams and implementing a post-processing correction algorithm to correct old mistakes. Additionally, filters that are oriented to filtering out predictable noise can be used to improve speech-recognition capabilities. While this may have an impact in performance, given the computational abilities present in 2021 compared to its creation in the 1990s may prove such improvements wouldn't reduce performance by a significant amount.

References

[1] Shmyrev, N. (2021). Building an application WITH POCKETSPHINX. CMUSphinx Open Source Speech Recognition. https://cmusphinx.github.io/wiki/tutorialpocketsphinx/. http://www.speech.cs.cmu.edu/tools/lmtool.html

[2] Shmyrev, N. (2021). General information. Speech at CMU. http://www.speech.cs.cmu.edu/tools/FAQ.html.

[3] Veisi, H., & Sameti, H. (2008). Improving the performance of speech recognition systems using fault-tolerant techniques. *2008 9th International Conference on Signal Processing*. https://doi.org/10.1109/icosp.2008.4697199

[4] Vargas, F., Fagundes, R. D., & Barros, D. (2001). Summarizing a new approach to Design speech recognition systems: A reliable NOISE-IMMUNE HW-SW version. *Symposium on Integrated Circuits and Systems Design*. https://doi.org/10.1109/sbcci.2001.953012

[5] Shmyrev, N. (2021). *Basic concepts of speech recognition*. CMUSphinx Open Source Speech Recognition. https://cmusphinx.github.io/wiki/tutorialconcepts/.

[6] Bloedorn, E., Michalski, R. Data-driven constructive induction: a methodology and its applications. IEEE Intelligent Systems, Special issue on Feature Transformation and Subset Selection, pp. 30-37, March/April, 1998

[7] Shmyrev, N. (2021). Sphinx Knowledge Base Tool at CMU. http://www.speech.cs.cmu.edu/tools/lmtool-new.html.

[8] Jurafsky , D., & Martin, J. H. (2020). N-gram Language Models. In Speech and Language Processing (pp. 1–26). essay, Stanford University.

[9] Hrinchuk, O., Popova, M., & Ginsburg, B. (2020). Correction of automatic speech recognition with transformer sequence-to-sequence model. ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). https://doi.org/10.1109/icassp40776.2020.9053051

[10] Hermus, K., Wambacq, P., & Van hamme, H. (2006). A review of signal Subspace SPEECH enhancement and its application to Noise robust speech recognition. *EURASIP Journal on Advances in Signal Processing*, *2007*(1). https://doi.org/10.1155/2007/45821