Lightweight domain adaptation: A filtering pipeline to improve accuracy of an Automatic Speech Recognition (ASR) engine

Jordan Hosier
Vail Systems, Inc.
Chicago, IL, USA
jhosier@vailsys.com

Yu Zhou
Vail Systems, Inc.
Chicago, IL, USA
yzhou@vailsys.com

Nikhita Sharma
nsharma@vailsys.com
Vail Systems, Inc.
Chicago, IL, USA

Vijay K. Gurbani
vgurbani@vailsys.com
Vail Systems, Inc.
Chicago, IL, USA

ABSTRACT

Transformer models have accelerated the field of speech recognition; deriving a low word error rate (WER) is demonstrably achievable under varying conditions. However, most ASR engines are trained on acoustic and language models constructed from corpora that include news feeds, books, and blogs in order to demonstrate generalization, leading to errors when the model is applied to a specific domain. While the increase in WER is acute for very specific domains (health and medicine), our work shows that it is sizable even when the domain is general (hospitality). For such domains, a *lightweight* adaptation approach can help; *lightweight* because the adaptation does not require extensive post-hoc training of additional domain-specific acoustic- or language-models that act as adjutants to the base ASR engine. We present our work on such *lightweight* filtering pipeline that seamlessly integrates lightweight models ($n-$gram, decision trees) with powerful, pre-trained, bi-directional transformer models, all working in conjunction to derive a 1-best hypothesis word selection algorithm. Our pipeline reduces the WER between 1.6% to 2.5% absolute while treating the ASR engine as a black box, and without requiring additional complex discriminative training.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms**; **Speech recognition**.

KEYWORDS

ASR, domain adaptation, decision trees, BERT, filtering pipeline, transformer architecture

ACM Reference Format:
Jordan Hosier, Yu Zhou, Nikhita Sharma, and Vijay K. Gurbani. 2021. Lightweight domain adaptation: A filtering pipeline to improve accuracy of an Automatic Speech Recognition (ASR) engine. In *2021 4th International Conference on Algorithms, Computing and Artificial Intelligence (ACAI'21), December 22–24, 2021, Sanya, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3508546.3508641

ACAI'21, December 22–24, 2021, Sanya, China

ACM ISBN 978-1-4503-8505-3/21/12…$15.00
https://doi.org/10.1145/3508546.3508641

1 INTRODUCTION

Despite the widespread availability of Automatic Speech Recognition (ASR) systems in both open source and commercial applications, high error rates in some domains remain a lingering impediment for effective speech recognition. The effect of these errors on the overall performance of ASR systems highlights the need for techniques to automatically detect and rectify such errors. This is important not only for improving overall ASR performance, but also to contain potential adverse effects of such errors on downstream language modeling processes and post-hoc analyses. The standard metric of ASR evaluation is WER (lower values preferred). Some ASR systems have been found to be successful under well controlled, ideal circumstances, with WER in the 5-10% range [29]. However, WER can increase substantially in noisy, unfamiliar settings, particularly Large Vocabulary Continuous Speech Recognition (LVCSR) applications [8]. The gap in WER between an ideal deployment and LVCSR applications motivates alternative methods to independently mitigate ASR transcription errors. Such a system should be capable of, independent of the ASR system used, further minimizing the overall WER.

Contribution

We seek to identify and resolve mis-transcriptions present in ASR transcripts. These transcripts are comprised of brief customer satisfaction surveys recorded by a customer at the end of a call center interaction. The recorded surveys are presented to Kaldi ASR, an open-source toolkit for speech recognition [19], which produces corresponding text transcripts of the surveys. Our novel error detection and correction filtering pipeline, which we present next, has the following advantages:

1) *Lightweight in its domain adaptation*:

Domain adaptation is most effective when a general purpose ASR engine is used in a niche domain, healthcare and medicine, for instance. Because a general purpose ASR engine does not have a language model tuned to recognize the words and utterances used in that domain, it is not uncommon to see a WER decrease of of 5-7% [13] when domain adaptation is used in highly domain-specific settings. However, our interest is in *lightweight domain adaptation*;

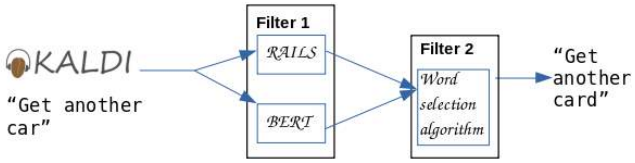

Figure 1: Lightweight filtering pipeline

We consider domain adaptation as lightweight when a general purpose ASR engine, with its native acoustic and language model, will suffice, but due to a lossy channel (mobile phone), or a noisy channel (random background noise), the ASR system is not able to perform optimally. As an example, consider the phrase "get another car." In banking survey transcripts, it is more likely that the customer said, "get another card" instead, but due to a degraded channel signal the ASR errs on the final phoneme, /d/, in the word "card."

2) *Lighweight in its approach*: Existing literature (Section 2) indicates that techniques to improve the ASR engine accuracy often entail additional training of deep neural acoustic models and bespoke language models.

These ancillary models encompass domain-specific artifacts, and are used as adjutants to the baseline general purpose ASR model to improve performance. Our approach does not require such adjutants, the generation of which is a time consuming process. Instead, it uses a model which is built on simple *n*-grams and phonetic encodings and seamlessly integrates a pre-trained unsupervised universal natural language model, BERT (Bidirectional Encoder Representations from Transformers [6]).

3) *Automatic operations*: Our approach does not require the user to act as an arbiter to improve accuracy. Instead, the word selection algorithm automatically performs operations to improve accuracy in the best case, and in the worst case, does not decrease the accuracy from its baseline.

4) *Uses simpler ASR-decoder features*: Our approach eschews more complex discriminative training of acoustic and language models in favor of simpler ASR-decoder features, i.e., features generated as a by product of the ASR process, namely, a word confidence score. We treat the ASR system as a black box and only require it to produce word confidence scores as part of its decoding output.

Our novel filtering pipeline is shown in Figure 1. The first filter in the pipeline receives the generated transcript from Kaldi, annotated with word confidence. This filter is comprised of two models: (1) RAILS model (named after our research group), a lightweight domain-specific model constructed from a corpus using *n*-gram probabilities, augmented with a phonetic distance calculation (cf., Section 3.5.1), and (2) BERT, a pre-trained bi-directional transformer-based language representation model (cf., Section 3.5.2). Each model sends its output to the second filter, which fits these to a trained decision tree model. The second filter synthesizes the final output automatically, i.e., without any human intervention (cf. Section 3.5.3).

The result of the filtering pipeline is more accurate transcripts via the automatic detection and correction of ASR errors. An example of such errors, and the pipeline's ability to rectify them, is visualized in Table 1.

We apply two filtering approaches on transcriptions generated from three pre-trained Kaldi ASR models: Api.Ai, Librispeech, and Zamia. The out-of-the-box Api.Ai model has an overall WER of 0.49 on our test dataset, consisting of call survey transcripts, while Kaldi's Librispeech model has a word error rate of 0.47. The Zamia model transcripts saw the lowest WER of 0.317. Despite the drop seen in the Zamia model, this WER is still far from the competitive WER's seen under ideal conditions and among commercial ASR systems.

The remainder of the paper is organized as follows: Section 2 reviews lines of research related to this effort. Section 3 presents the proposed filtering pipeline, followed by Section 4, where we present our results and a discussion of our approach. Finally, in Section 5, we provide concluding remarks and directions for future work.

2 RELATED WORK

Performance of ASR engines has improved tremendously with deep learning, getting closer to human-like WERs. By some measures [22, 29], the best human accuracies on well known datasets such as the Switchboard (SWB) and CallHome (CH) are 5.1% and 6.8%, respectively. Saon et al. [22] demonstrated a WER of 5.5% and 10.3% on SWB and CH, respectively, while Xiong et al. [29] claim 5.8% and 11.0%, respectively. While ASR systems continue to improve, there is still a wide variance in the WER rates among ASR engines and even within a single ASR engine depending on the particular domain. Traditionally, accuracy improvements in ASR engines have been driven by improvements in acoustic and language modeling, the availability of a large amount of training data, and high computational resources [4, 12, 28]. Because of the large corpora required in training ASR engines — thousands of hours to tens of thousands of hours, and even more — techniques have been developed to investigate speeding up the training phase. These techniques range from distributed training across high-performance computing infrastructure to bespoke hardware[1].

Despite these impressive advances, ASR errors remain inevitable. Some of the errors stem from the imperfection of today's speech recognition technology, while others are introduced due to inherent and irreducible noise in the out-of-sample (test) audio samples. Error detection can be done on decoder-based features, i.e., features generated as a byproduct of the ASR process – word confidence scores, language model, and similar artifacts – or be performed on the combination of decoder features and exogenous, non-decoder based features. Furthermore, before an error can be corrected, it must be detected; thus error correction techniques subsume error detection. Finally, once an error is detected, it can be corrected automatically, or require manual intervention. Within this taxonomy, our work is placed in the decoder-based, automatic (i.e., non-manual) error correction techniques. The majority of existing work focuses on error detection and manual error correction [1, 8, 18, 26]; below we place our work in the context of existing literature in decoder-based error correction techniques, specifically correction techniques that do not require human intervention.

[1]Nvidia's A100 Graphical Processing Unit (GPU) has more than 54 billion transistors, allowing a 20-fold increase in horsepower over the company's previous V100 GPU.

Table 1: Examples of corrected and nearly corrected samples, with the error in question in bold font.

Ground-Truth Sentence	ASR	Filtering Pipeline
Short sweet **and to** the point	Short sweet **into** the point	Short sweet **and to** the point
He was polite but that's not my **issue**	He was polite but that's not my **essay**	He was polite but that's not my **fault**
Very friendly and **helpful**	Very friendly and **info**	Very friendly and **helpful**

Sarma et al. [23] are motivated by searching regions in a transcript given a query word. They perform a lexical co-occurrence analysis using a large corpus to identify regions in the data that may contain likely context for a query word. Setlur et al. [24] perform error correction by flipping the hypotheses produced by an N-best recognizer in cases when the top candidate has a confidence score that is lower than that of the next candidate. Using a hybrid confidence measure, they reduced the WER by 0.13% on a restricted digit recognition task. Similarly, Zhou et al. [31] decreased their error rate by 0.8% by devising a linear scoring approach to score N-best (N=20) words that could be substituted for an erroneous word. Liu et al. [11] do not differentiate whether a certain word is right or wrong, but propose a score to evaluate to what degree the candidate word is appropriate in the hypothesis sentence based on topic modeling and word embedding, two semantic features. They reduce the WER between 0.29% to 0.51%.

More recent literature [7, 9, 13, 17, 25] suggests training out-of-domain machine translation models to better the results of a general ASR engine. The out-of-domain models are fairly complex, composed of hierarchical bi-directional LSTM (BLSTM) encoder and decoders that learn domain-specific utterances to improve on the base WER. When such trained out-of-domain models are used in niche applications (medical transcripts, where a general ASR engine will exhibit high WER), they tend to produce 5-7% WER improvements, while in other cases these complex models yield 3-4% WER improvements. Shivakumar et al. [25] present a Noisy-Clean Phrase Context Model that uses additional trained neural language models to reduce WER by 0.29% and 0.55% on two datasets. Bassil et al. [2] propose post-hoc filtering of transcripts using only an external n-gram dataset curated from Microsoft's Bing search engine.

By contrast, our filtering approach is a lightweight domain adaptation technique that only requires a simple domain-adapted n-gram model [15] and a pre-trained and untuned BERT model. We eschew more complex discriminative training of LVCSR tasks [21, 27] in favor of simple ASR-decoder features and lightweight learning models (n-grams and decision trees). Our proposed technique does not require additional training of neural network models, instead, using a pipeline approach, filters the output of the general purpose ASR engine through the pipeline to exhibit an average reduction of between 1.6% to 2.5% absolute WER across different pre-trained Kaldi models.

3 FILTERING PIPELINE

3.1 Background on ASR engines

The goal of an Automatic Speech Recognition system is to map an audio signal to its corresponding text. ASR systems generally consist of an acoustic model, a language model, and a decoder. ASR

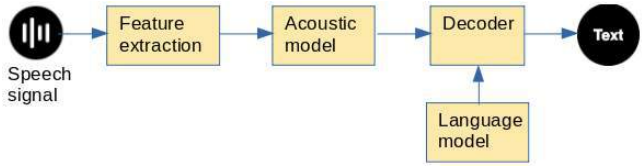

Figure 2: General framework of ASR systems

systems decode speech input to generate the best guess transcription of the words that were spoken, as text output. Briefly, the feature extraction module extracts representative features from the speech signal while suppressing unnecessary noise. The acoustic model is responsible for modelling the speech acoustics, and transcribing the extracted audio features into a sequence of context dependent phonemes. (Most acoustic models today are trained using deep neural networks.) The language model determines which words or sequence of words are more likely, given the surrounding words as context, and finally, the decoder uses the acoustic model, the grammar, and the language model collectively to generate word sequences that are likely for a particular audio frame. The word sequence with the highest probability is the final text output. The interested reader is directed to [30] for a detailed treatment of ASR systems.

First, the input waveform is split into small parts or frames, usually 25ms in length and some features are extracted from it. It is standard practice to extract MFCCs (Mel-Frequency Cepstral Coefficients), CMVNs (Cepstral Mean and Variance Normalization) that represent the content of the audio, or i-Vectors that capture the style of the speaker or utterance. It is essential that the features chosen suffice to identify the features of human speech while suppressing unnecessary noise. Thus, the audio signal is compressed into a sequence of fixed length vectors via feature extraction. Next, the acoustic model predicts which phoneme is being spoken in each frame of audio. The acoustic model is responsible for modelling the speech acoustics, transcribing the extracted audio features into some sequence of context dependent phonemes. Acoustic models are trained using Deep Neural Networks (DNNs) on large datasets, typically thousands of hours of human transcribed audio data.

A lexicon or dictionary maps each word to its phonetic representation, and is used in an ASR system to convert the predicted phonemes into words and eventually complete sentences. The language model determines which words or sequence of words are more likely, given the surrounding words as context. This context is typically derived from neural networks or n-gram models trained on a very large textual dataset. Finally, the decoder uses the acoustic model, the grammar, and the language model collectively to generate word sequences that are likely for a particular audio frame. The word sequence with the highest probability is the final text output.

Table 2: Comparison of pre-trained models

	Api.Ai	LibriSpeech	Zamia
Training data (hrs)	Api.Ai logs, LibriSpeech corpus (unknown hours)	Audio books (960 hours)	Various (1,200 hours)
Type of model	nnet3 chain	TDNN chain	TDNN chain
Size	177 MB	2 GB	609 MB
Words in vocabulary	128K	200K	169K

Table 3: Dev/test split.

	Dev		Test	
Model Data	Audio files	Words	Audio files	Words
Api.Ai	887	279,156	813	256,036
LibriSpeech	888	281,214	812	257,374
Zamia	888	278,553	812	274,938

3.2 Acoustic and language models used

To evaluate our novel filtering pipeline, we use three pre-trained Kaldi models: an Api.Ai model[2], a LibriSpeech model[3], both provided as models in the Kaldi distribution. The third model is provided by Zamia[4], a third party under an open source license. The Api.Ai model is the simplest of the models, trained on a mix consisting of English audiobooks and short commands ("Wake me up at 7am."). The LibriSpeech model, consisting of a chain system based on Time Delay Neural Net (TDNN), consists of approximately 1,000 hours of audiobook content, carefully segmented and aligned. The third model from Zamia is the most resource intensive of the models, providing the best accuracy by being trained on about 1,200 hours of high quality, noise resistant audio. Like LibriSpeech, the Zamia model is also a large nnet3-chain factorized TDNN model. (See Table 2 for a comparison.)

3.3 The Data

Subsequent analyses are derived from customer call surveys; the original audio files consisted of customers leaving voice survey responses about their experience with the call center agent. The surveys varied in length from one word to approximately 300 words. The data was given as input to each of the three Kaldi models in the form of 1,700 audio files transcribed by human transcribers. These transcriptions serve as the ground truth, a benchmark to evaluate the word error rate (WER) of both the raw Kaldi output and the adjusted output after applying our filtering pipeline.

The WER measures the percentage of incorrect words by adding up the total number of Substitutions (S), Insertions (I), and Deletions (D) that occur in a sequence of recognized words, which is determined by aligning any automatically generated transcriptions with the ground-truth, human transcriptions. The sum is subsequently divided by the total number of words (N), resulting in the WER as shown in Equation 1.

$$WER = \frac{S + D + I}{N} \tag{1}$$

The filtering pipeline is developed and tested on a development/test split of Kaldi generated transcript datasets derived from the three pre-trained Kaldi models: Api.Ai, LibriSpeech, and Zamia (Table 3). Since each model makes different deletion and insertion errors, the number of words in the development and test sets are slightly different across the Kaldi models. The development sets are used to tune the parameters of the RAILS model and the word selection algorithm.

3.4 Preliminaries

As Figure 1 demonstrated, our lightweight filtering pipeline works by examining the output of Kaldi, finding regions of low confidence (i.e. words). These regions are subsequently presented to the first filter, which suggests improvements on low confidence words. These improvements are sent to the second filter, which in turn uses a word selection algorithm to synthesize the suggestions and produces a 1-best replacement hypothesis for the Kaldi low confidence word.

To aid in subsequent discussion, we formalize the definitions that appear in the rest of the paper. Let K be the Kaldi region that exhibits low confidence, and let $C(K)$ be a function that computes the confidence of K:

$$C(K) \in \{x, -1\}, x \in \mathbb{R} : 0.0 \leq x \leq 1.0 \tag{2}$$

i.e., $C(K)$ is either x or -1, where x represents the confidence of Kaldi in the rendering region K. This confidence is calculated based on posterior probabilities in the decoding lattice. One exception to this is in the case of deleted regions that we represent as -1. During training, deleted words can be identified after aligning the ASR generated transcripts with ground-truth human transcripts. After alignment leads to the identification of deleted words, we automatically mark them with -1.

In order to improve K, the RAILS model creates $R = \{R_1, ..., R_N\}$, an ordered N-best hypotheses set, where $N <= 10$. The algorithmic filter is provided R and using BERT, it produces $B = \{B_1, ..., B_N\}$, an ordered N-best hypotheses set ($N = 10$). For both R and B, the first element in the set, R_1 and B_1, respectively, is of interest as this element is the hypothesis that represents the replacement region. Additionally, the original region, K, also remains a candidate hypothesis. Therefore, let $F \in \{R_1, B_1, K\}$ refer to this final hypothesis that serves as a replacement region. (Note that $\{R_1, B_1, K\}$ is a multiset since it may contain multiple instances of the same element.)

To evaluate the goodness of hypothesis F, an updated confidence function $C'(F)$ is defined:

$$C'(F) = \begin{cases} -1 : & C(F) = \{-1\} \\ C(F) \setminus \{-1\} : & \text{otherwise} \end{cases} \tag{3}$$

where $C(F)$ is defined in Equation 2. Equation 3 enumerates three potential outcomes of the pipeline applied to K, the region with low confidence:

[2]https://github.com/dialogflow/api-ai-english- asr-model, last visited Apr 16, 2021.
[3]https://kaldi-asr.org/models/m13, last visited Apr 16, 2021.
[4]https://goofy.zamia.org/zamia-speech/asr-models/kaldi-generic-en-tdnn_fl-r20190609.tar.xz, last visited Apr 16, 2021.

Table 4: Training specifications.

Audio files	Words
86,924	1,816,970

1. $C'(F) = C'(K)$: Algorithm chooses $F = K$ as the final hypothesis for the replacement region. In such an event, WER remains unchanged.
2. $C'(F) > C'(K)$: Algorithm chooses $F \in \{R_1, B_1, K\} \setminus \{K\}$, i.e., it chooses either R_1 or B_1 as the final hypothesis for the replacement region. This choice is correct, thus decreasing the WER.
3. $C'(F) < C'(K)$: Algorithm chooses $F \in \{R_1, B_1, K\} \setminus \{K\}$, i.e., it chooses either R_1 or B_1 as the final hypothesis for the replacement region. This choice is incorrect because K was actually rendered correctly and the final replacement word suggested by the pipeline is erroneously applied, increasing WER.

3.5 The Filters

3.5.1 Filter 1: RAILS Model. The RAILS model is a real-time, automatic instantiation of the approach presented in [10]. Given low-confidence word, the model suggests higher probability replacements. These probabilities are generated from n-grams trained on a corpus of approximately 87,000 customer survey transcripts consisting of 1.8 million words (See Table 4). The model is constructed from the following building blocks: (1) a dataset containing all uni-, bi- and tri- and four-grams from the corpus and their respective probabilities, and (2) a phonetic encoding component, known as the match rating approach (MRA), which is used to determine phonetic similarity between lexical items [16].

The RAILS model is applied to words with low confidence and if the context is sufficient to make an alternate prediction, the model uses n-grams and phonetic distance measures to suggests an N-best list, $R = \{R_1, ..., R_N\}, (N \leq 10)$. R may be null in the absence of context among the trained n-grams. When R is not null, elements in the list are sorted by their probability, i.e., the most preferred element is in R_1.

3.5.2 Filter 1: BERT. BERT (Bidirectional Encoder Representations from Transformers) [6] is a language representation model using a Transformer network trained on large text corpora. It has achieved state-of-the-art performance on various NLP tasks. Processing an input text with certain words masked out, it can make predictions on the masked words by extracting contextual information from all surrounding text. In this work we use BERT to complement the RAILS model result. Using the pre-trained BERT Large Masked Language Model (MLM) without fine-tuning, for each low confidence word K in Kaldi transcript, an ordered N-best hypothesis set $B = \{B_1, ..., B_N\}, (N = 10)$ is generated as potential replacement candidate. Like the RAILS model, elements in B are sorted by probability in context, i.e., the highest probability element is in B_1.

3.5.3 Filter 2: A word selection algorithm. For a given low confidence word K in a Kaldi transcript, if both ordered hypothesis sets from the RAILS model, R, and BERT, B, are available, then it is possible to select a final hypothesis (word) F in place of K that results in a higher probability of matching the ground truth (human transcription) than always using R_1 or B_1 alone. Therefore the problem is reduced to specifying a word selection algorithm to choose $F \in \{R_1, B_1, K\}$ in place of K to minimize WER.

This word selection algorithm is developed with the assistance of R and B, and their intersection $T = R \cap B = \{T_1, ..., T_n\}, (n \leq N)$, with the same element ordering as R. We chose the set T to add more variance and to better generalize the decision trees. To motivate the intuition behind this, consider $R = \{a, b, c, f\}$, and $B = \{x, b, y, f\}$; then, $T = \{b, f\}$. Clearly, $T_1 = b$ is considered as a high probability candidate by both RAILS and BERT models, thus allowing it as a possible response seems appropriate. In a sense, T_1 acts as an arbiter to further increase the confidence that F has a high probability of being correct. To observe this, note that $F \in \{R_1, B_1, K\}$ implies that if K is not chosen as the final response, then there is no reliable arbiter to determine which of R_1 or B_1 should be chosen (c.f. outcomes 2 and 3 of Eq. 3). Elements in T have a high probability of being correct since they appear in both lists, R and B. Thus, T_1 can be considered an arbiter by observing which of B_1 or R_1 matches it, and then considering the matched element to be the final response.

Algorithm 1: Word selection algorithm

```
for all utterances do
    while low confidence regions in utterance do
        K ← region of low confidence
        R ← RAILS(K) // RAILS suggestion
        B ← BERT(K) // BERT suggestion
        T ← R ∩ B
        F ← Predict_Response(R,B,K,T)
    end
end
Function Predict_Response(R, B, K, T):
    if K = ∅ then
        /* Deletion                                   */
        if (R_1 = ∅) or (B_1 ∈ R and B_1 ≠ T_1) then
            F ← B_1
        else
            F ← R_1
        end
    else
        /* Substitution                               */
        if (R_1 ∈ T) and (K ≠ B_1) then
            F ← R_1
        else
            F ← K
        end
    end
    return F
```

The word selection algorithm is extracted from a decision tree model [20] and shown in Algorithm 1. We prefer decision trees to other hypothesis sets for their interpretability and intuition in understanding how a particular decision was made. We created a multi-class decision tree that predicts the class of $F \in \{R_1, B_1, K\}$. To train the decision tree model, all possible Boolean features belonging to the categories of word validation, word comparison, and word membership are utilized. There are 15 features total, they are shown in Table 5.

Table 5: Boolean features for Decision Tree

Word Validation	Word Comparison	Word Membership
$K = \varnothing$	$K = R_1$	$K \in R$
$R_1 = \varnothing$	$K = B_1$	$K \in B$
	$R_1 = B_1$	$R_1 \in B$
	$K = T_1$	$B_1 \in R$
	$R_1 = T_1$	$K \in T$
	$B_1 = T_1$	$R_1 \in T$
		$B_1 \in T$

Table 6: Performance of the RAILS model in Filter 1.

ASR Model	Filter Model	WER	Corr-ected	Mis-corrected	Net Corrected
Api.Ai (Base WER:0.492)	RAILS	0.481	539	411	128
Librispeech (Base WER: 0.443)	RAILS	0.436	380	301	79
Zamia (Base WER: 0.317)	RAILS	0.303	307	69	238

Word validation determines if K and R_1 are null. (Note that there is no check for $B_1 = \varnothing$ because unlike the RAILS model that can return $R = \{\varnothing\}$ if it does not find the appropriate context in the n-grams, BERT always returns a non-null N-best list, B.) Word comparison creates features by exhaustively exploring the symmetric pairings of each element in $\{R_1, B_1, K\}$, with an additional three features that capture the pairing of $\{R_1, B_1, K\}$ with T_1. Word membership creates binary features by examining the membership of the response variables $\{R_1, B_1, K\}$ in the sets R, B, and T. Human transcription is used as the ground truth for training the decision tree, and a missing word in Kaldi transcription is represented as NULL ($\varnothing$).

To train the decision tree, first the dataset is partitioned into dev and test (Table 3). Each audio file in the dev partition is considered an utterance and is examined for low confidence words. Each low confidence word becomes a new training (and test) vector that contains the 15 attributes in Table 5 and the response variable. Thus, one utterance may generate multiple training (and test) vectors. The trained tree is then examined, the algorithm shown in Algorithm 1 extracted, subsequently used in Filter 2 (cf. Section 3.5.3). The results from the algorithm fitted on the test set are discussed in Section 4.1.3.

4 EVALUATION RESULTS AND DISCUSSION

We now present the results and subsequent discussion on applying our pipeline. We first present the results of the RAILS model and the BERT model on the datasets to observe their effects when used in isolation before introducing a decision-tree based word selection algorithm.

4.1 Performance

4.1.1 Filter 1: RAILS model performance. The Kaldi Api.Ai model saw a 1.1% improvement in overall WER when only the RAILS model was applied (Table 6). This improvement was largely a result of bringing the deletion rate down by 8%. The substitution rate, however, went up. There were instances where the RAILS model was applied on **correct** Kaldi transcriptions with low word confidence, which resulted in a higher substitution WER. The RAILS model is applied to all words with a confidence of 0.60 or less, the optimal threshold found for this model data. This threshold was optimized to maximize the number of correct replacements and minimize the number of incorrect replacements of words with low word confidence.

Of all words below this threshold on which the RAILS model is applied, some are correctly changed, some are incorrectly changed, and some remain unchanged if the context is not sufficient to suggest an alternative. Thus, while we see 8% gains in the deletion rate, this is partially negated by the increase in substitution error rates, or mis-corrections. Further, some increases in the substitution rate can be attributed to deletion errors being moved to substitution errors in cases where the RAILS model hypothesis for the deleted word in the Kaldi transcript does not match the ground truth.

The LibriSpeech model saw a .07% improvement in overall WER after application of the RAILS model (Table 6). As was the case in the Api.Ai model, the substitution rate increased. In this model data, the RAILS model is applied to all words with a confidence of 0.50 or less, the threshold found to be optimal for this model data.

The Zamia model saw a 1.4% absolute improvement in overall WER after application of the RAILS model (See Table 6). Under this regime, the RAILS model is applied to all words with a confidence of 0.40 or less, the optimal threshold for this data. Once again, we see that gains in WER are limited by the rate of mis-substitutions. For this reason, we use BERT as an additional model in an effort to decrease the rate of mis-corrections.

Table 7: Results appended with BERT.

ASR Model	Filter Model	WER	Corr-ected	Mis-corrected	Net Corrected
Api.Ai (Base WER: 0.492)	RAILS	0.481	539	411	128
	BERT	0.494	397	526	-129
Librispeech (Base WER: 0.443)	RAILS	0.436	380	301	79
	BERT	0.450	255	381	-126
Zamia (Base WER: 0.317)	RAILS	0.303	307	69	238
	BERT	0.311	197	99	98

4.1.2 Filter 1: BERT performance. The pre-trained BERT model is used without fine-tuning on our dataset, therefore it is not as accurate as the RAILS model. The results of using BERT as the only model in filter 1 are shown in Figure 7. Compared to using the RAILS model, BERT demonstrates a consistent increase in WER when we fit our data with BERT. To this end, BERT in a stand alone fashion can serve as a benchmark for the RAILS filter. The more interesting aspect of BERT's performance — and an explanation of the increase in WER — is shown in the last column of Table 7. For the less complex ASR models (Api.Ai and LibriSpeech), BERT's net corrected is negative, while for the complex Zamia ASR model it is positive. This would appear to imply that BERT works best when the neighboring words are contextually accurate. Because

Table 8: Results appended with decision tree models.

ASR Model	Filter Model	WER	Corr-ected	Mis-corrected	Net Corrected
Api.Ai (Base WER:0.492)	RAILS	0.481	539	411	128
	BERT	0.494	397	526	-129
	DT-A	0.467	503	52	451
Librispeech (Base WER: 0.443)	RAILS	0.436	380	301	79
	BERT	0.450	255	381	-126
	DT-L	0.427	309	25	284
Zamia (Base WER: 0.317)	RAILS	0.303	307	69	238
	BERT	0.311	197	99	98
	DT-Z	0.300	325	9	316

the less complex ASR models make more mistakes on the average, they deprive BERT from the context needed to correct errors.

Finally, for the sake of completeness we note that the results in Table 7 are obtained using the same word confidence thresholds that have been optimized for achieving the best RAILS model results (c.f., Section 4.1). However, we note that these thresholds do not affect the results from BERT in any quantitative manner since the purpose-trained RAILS model will be more effective for this application than the pre-trained BERT model. The contribution of BERT becomes apparent as we move to filter 2, and this is discussed next.

4.1.3 Filter 2: Word selection algorithm performance. The benefit of the pipeline lies in the processing of Filter 2. In this filter, the N-best list from BERT is used in combination with the N-best list from the RAILS model. The algorithms that perform these combinations and select a word are derived from training a decision tree model as discussed in Section 3.5. Three decision tree models are induced — one for each ASR model — on the dev set and evaluated on the test set (Table 3). The results from evaluating the decision trees on the test set in terms of WER and associated information are summarized in Table 8. (In the table, DT-A, DT-L and DT-Z refer to the learned decision trees for Api.Ai, LibriSpeech and Zamia models, respectively.) Next, we analyze the results for the Api.Ai model, analysis of the other two ASR models will follow similar reasoning.

Table 8 shows that while the RAILS model is able to correct 539 low-confidence regions in the Api.Ai model, it also made 411 errors. These errors are a result of the originally correct transcription word (K) being replaced by incorrect RAILS model 1-best hypothesis (R_1) in regions of low confidence. The selection function induced by training a decision tree (indicated by the row DT-A), demonstrates that the mis-corrections were reduced by a factor of 8 (or 800%) when compared to the mis-corrections made by the purpose-trained RAILS model. Using a decision tree word selection algorithm brings the mis-corrections down to 52 (from 411). This also has the serendipitous side effect of increasing the net corrected by approximately 350% (or a factor of 3.5) compared to the bespoke RAILS model. The net result of the application of filter 2 is that the pipeline has decreased the WER from the Api.Ai model base WER of 0.492 to 0.467 when using the decision tree word selection algorithm, a decrease of 0.025 (or 2.5%) in absolute terms.

Table 9: Decision tree accuracy for ASR Models.

	Low confidence regions	Decision tree matches ground truth	Decision tree accuracy (%)
Api.Ai	7,888	1,677	21.3
LibriSpeech	5,540	1,296	23.4
Zamia	3,375	563	16.7

Using similar reasoning, the decision tree word selection algorithm for the LibriSpeech ASR model reduced the mis-corrections by a factor of 12 (or 1200%, from 301 mis-corrected when using RAILS model to only 25 mis-corrected when using DT-L), and the WER decreased by 0.016 (or 1.6% absolute, from a base of 0.443 to 0.427). The decision tree word selection algorithm for the Zamia ASR model reduced the mis-corrections by a factor of 7 (or 700%, from 69 mis-corrected when using RAILS model to only 9 mis-corrected when using DT-Z), and the WER decreased by 0.017 (or 1.7% absolute, from a base of 0.317 to 0.300) when using DT-Z.

A second observation is apparent in the WER decrease as the ASR models become more complex. The Zamia ASR model, the most complex of the three, has a WER of 0.317. However, even at this WER, our filtering pipeline was able to further decrease the WER to 0.300 when using the decision tree selection function, a decrease of 0.017 (or 1.7%) absolute.

4.2 Discussion: Using a decision tree as a word selection function

The WER for each ASR model is a function of the complexity and size of the model's training. The more complex the model, the less error it makes leading to a smaller WER as shown in Table 8. Consequently, for each of the three ASR models, the distribution represented by the regions of low confidence are different, and we treat each such distribution as an independent dataset on which to train a decision tree.

As discussed in Section 4.1.3, the decision trees are able to reduce the WER between 1.6% to 2.5% absolute, depending on the ASR model. For the Api.Ai model, the decision tree is able to reduce the mis-corrections by 800%. Thus, clearly, the decision trees contribute to the reduction of the WER and to the associated increase in the accuracy of the generated transcript. However, if we were to examine the traditional performance measures associated with decision trees (accuracy, precision, and recall), we discover that they do not quite reflect the gains in performance shown in Section 4.1.3. Table 9 demonstrates an accuracy of only 21.3% for the Api.Ai model for decision tree model induced on the dev partition and fit to the test partition.

Clearly, the table shows that traditional performance measures alone are not very high. However, just as clearly, the discussion in Section 4.1.3 demonstrates that the decision trees have a large impact on reducing the mis-corrections, as well as the overall reduction in the WER in our pipeline. Next, we discuss the reasons why the traditional performance of decision trees as measured in Table 9 is low; this discussion will motivate the need for proposing a new performance measure we call *predictive utility* to evaluate

Table 10: Measuring the impact of decision trees.

	RAILS or BERT or KALDI matches ground truth	Decision Tree matches ground truth	Predictive utility (P)
Api.Ai	2,048	1,677	81.9%
LibriSpeech	1,556	1,296	83.3%
Zamia	687	563	82.0%

the contribution of decision tree as a word selection algorithm in our work.

Recall from the discussion in Section 3.5.3 that 15 binary features were extracted from the training dataset, and decision trees are learned. The shape of the training data is Mx15, where M is 2067, 1585, and 734 for the Api.Ai, LibriSpeech, and Zamia models, respectively. Theoretically, for 15 binary features, a decision tree would need to be trained on $2^{15} = 32,768$ observations to learn all patterns. (However, note that even if we had access to such a training corpus, accuracy will be < 100% because for many observations, the response variable $F \in \{R_1, B_1, K\}$ will not match the human transcribed ground truth, i.e., neither $R_1, B_1,$ or K matches the ground truth.) In our case, the training datasets contains only between 2% to 6% of the theoretical number of observations from which to find patterns and generalize. The low number of training observations occur primarily because we can only use samples for training where one of $\{R_1, B_1, K\}$ matches the human transcribed ground truth. For most of the low confidence regions in the dev partion on which the trees are trained, none of $\{R_1, B_1, K\}$ matches the ground truth. Thus, with training occurring on an extremely small subset of observations, traditional decision tree performance measures (precision, recall, accuracy) do not suffice. Our aim is to extract enough residual information from predictions of the decision tree to aid in lowering the overall WER, for this, we formulate a measure called *predictive utility*. Eq. 4 quantifies the predictive utility (P, in percentage) of the decision trees:

$$P = \frac{|D|}{|A|} * 100. \tag{4}$$

Here, D is a multiset defined as follows: $D = \forall w \in \{\mathrm{L}\} : \{d_w | d_w = \mathrm{Truth}\}$, where L is a set of all low confidence regions (column 1 of Table 9), d_w is the predicted output of the decision tree on a low confidence region w, and d_w matches the ground truth (Truth).

In Eq. 4, A is a multiset defined as follows: $A = \forall w \in \{\mathrm{L}\} : \{a_w | a_w = \mathrm{Truth}\}$. In order for a_w to be included in A, *one* of the following conditions must hold:
1. $a_w = R_1 = \mathrm{Truth}$: a_w must be equal to the RAILS 1-best hypothesis for w, which must be equal to the ground truth; or
2. $a_w = B_1 = \mathrm{Truth}$: a_w must be equal to the BERT 1-best hypothesis for w, which must be equal to the ground truth; or
3. $a_w = K = \mathrm{Truth}$: a_w must be equal to K, the word chosen by Kaldi, and K must be equal to the ground truth.

Table 10 shows the predictive utility, P, for every ASR model. Without a decision tree selection algorithm to guide the choice of a specific element to choose from $\{R_1, B_1, K\}$, any one of these has an equal probability of being chosen. For the Api.Ai ASR model, there are 2,048 observations where one of these elements matches the ground truth, but it is not clear which one to choose. The decision tree is able to correctly predict 1,677 of these observations, leading to a predictive utility of 81.9%. Similar analysis applies for the remaining two ASR models. It is important to note that despite the complexity of the ASR model being used, the decision tree word selection algorithm improves the predictive utility by an average of 82.4%.

5 CONCLUSION AND FUTURE WORK

In this paper, we propose a lightweight filtering pipeline consisting of two filters: the first filter fits lightweight trained model (n-grams), and pre-trained, out-of-the-box language model (BERT) to the transcripts generated by a general purpose ASR engine. Filter 2 uses a trained decision tree to derive a word selection algorithm to minimizes both the mis-corrections, and the WER. Our algorithm makes the correct selection on up to 83.3% of the low-confidence words, resulting in a WER reduction of 1.6% to 2.5% absolute. Our pipeline does not require training deep learning models or human intervention. It takes advantage of advances in attention based transformer models by seamlessly integrating powerful, pre-trained models.

The benefits of our particular approach are as follows. First, our filtering pipeline does not require training any deep learning models nor does it require any human intervention. To the extent that additional training is required in our pipeline, that training is lightweight, namely, an n-gram model and decision trees. Second, our pipeline takes advantage of advances in attention based transformer models by seamlessly integrating powerful, pre-trained models like BERT. Third, our approach also treats the ASR engine as a black box, and only requires that it output a word confidence score as a part of the decoding to detect potential word errors. Thus, the filtering pipeline can be applied in an independent, post-hoc fashion. This approach is unique in the realm of domain-adaptation in that the ASR errors arise largely from a lossy or noisy channel, rather than errors resulting from deploying the ASR system on an out-of-domain vocabulary.

There are a number of areas of further exploration. First, as attention-based transformer models benefit from continued research, our pipeline can seamlessly integrate other pre-trained deep bi-direct-ional language models like Elmo [14] and GPT-3 [3]. We would like to characterize the efficacy of such integration. Next, the models in Filter 1 can be easily parallelized; each model independently creates an N-best hypotheses set from the transcript and sends its output to Filter 2, an architecture highly amenable to a map-reduce computing paradigm [5] where all maps are parallelized. A third area of exploration is investigating the optimal word confidence threshold for BERT and optimizing this hyperparameter for further contribute in Filter 2. In our current work, we learn the word threshold through a grid search while training the RAILS model, and reuse it for BERT. Finally, we treat the ASR engine as a black box, and only requires that it outputs a word confidence score while decoding. The filtering pipeline can be applied in an independent, post-hoc fashion to mitigate errors arising from lossy or noisy channels, a reality in any deployment.

REFERENCES

[1] W. A. Ainsworth and S. R. Pratt. 1992. Feedback Strategies for Error Correction in Speech Recognition Systems. *Int. J. Man-Mach. Stud.* 36, 6 (June 1992), 833–842.

[2] Youssef Bassil and Paul Semaan. 2012. ASR Context-Sensitive Error Correction Based on Microsoft N-Gram Dataset. *ArXiv* abs/1203.5262 (2012).

[3] Tom B. Brown et al. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

[4] Xiaodong Cui, Wei Zhang, Ulrich Finkler, George Saon, Michael Picheny, and David S. Kung. 2020. Distributed Training of Deep Neural Network Acoustic Models for Automatic Speech Recognition: A comparison of current training strategies. *IEEE Signal Process. Mag.* 37, 3 (2020), 39–49. https://doi.org/10.1109/MSP.2020.2969859

[5] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv e-prints*, Article arXiv:1810.04805 (Oct. 2018), arXiv:1810.04805 pages. arXiv:1810.04805 [cs.CL]

[7] Luis D'Haro and Rafael Banchs. 2016. Automatic Correction of ASR Outputs by Using Machine Translation. In *Interspeech*. 3469–3473.

[8] Rahhal Errattahi, Asmaa El Hannani, and Hassan Ouahmane. 2018. Automatic speech recognition errors detection and correction: A review. *Procedia Computer Science* 128 (2018), 32–37.

[9] Yohei Fusayasu, Katsuyuki Tanaka, Tetsuya Takiguchi, and Yasuo Ariki. 2015. Word-Error Correction of Continuous Speech Recognition Based on Normalized Relevance Distance. In *IJCAI*.

[10] Jordan Hosier, Vijay K Gurbani, and Neil Milstead. 2019. Disambiguation and Error Resolution in Call Transcripts. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 4602–4607.

[11] Chang Liu, Pengyuan Zhang, Ta Li, and Yonghong Yan. 2019. Semantic Features Based N-Best Rescoring Methods for Automatic Speech Recognition. *Applies Sciences* 9(23):5053 (2019).

[12] Yanhua Long, Yijie Li, Shuang Wei, Qiaozheng Zhang, and Chunxia Yang. 2019. Large-Scale Semi-Supervised Training in Deep Learning Acoustic Model for ASR. *IEEE Access* 7 (2019), 133615–133627.

[13] A. Mani et al. 2020. ASR Error Correction and Domain Adaptation Using Machine Translation. In *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*.

[14] Peters Matthew et al. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 2227–2237. https://doi.org/10.18653/v1/N18-1202

[15] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013).

[16] Gwendolyn B Moore et al. 1977. Accessing Individual Records from Personal Data Files Using Non-Unique Identifiers. Final Report. Computer Science & Technology Series. (1977).

[17] Ryohei Nakatani, Tetsuya Takiguchi, and Yasuo Ariki. 2013. Two-step correction of speech recognition errors based on n-gram and long contextual information. In *INTERSPEECH*.

[18] J. M. Noyes and C. R. Frankish. 1994. Errors and error correction in automatic speech recognition systems. *Ergonomics* 37, 11 (1994), 1943–1957.

[19] Daniel Povey et al. 2011. The Kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society.

[20] J. R. Quinlan. 1986. Induction of Decision Trees. *Mach. Learn.* 1, 1 (March 1986), 81–106.

[21] Brian Roark, Murat Saraclar, and Michael Collins. 2007. Discriminative n-gram language modeling. *Computer Speech & Language* 21, 2 (2007), 373 – 392. https://doi.org/10.1016/j.csl.2006.06.006

[22] George Saon et al. 2017. English Conversational Telephone Speech Recognition by Humans and Machines. In *Proc. Interspeech 2017*. 132–136.

[23] Arup Sarma et al. 2004. Context-Based Speech Recognition Error Detection and Correction. In *Proc. of HLT-NAACL 2004: Short Papers* (Boston, Massachusetts). Assn. for Computational Linguistics, 85–88.

[24] A. R. Setlur et al. 1996. Correcting recognition errors via discriminative utterance verification. In *Proc. of* 4^{th} *Intl. Conf. on Spoken Language Processing*., Vol. 2. 602–605.

[25] Prashanth Gurunath Shivakumar et al. 2019. Learning from past mistakes: improving automatic speech recognition output via noisy-clean phrase context modeling. *APSIPA Trans. on Signal and Information Processing* 8 (2019).

[26] Y. Tam et al. 2014. ASR error detection using recurrent neural network language model and complementary ASR. In *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing*.

[27] P.C. Woodland and D. Povey. 2002. Large scale discriminative training of hidden Markov models for speech recognition. *Computer Speech & Language* 16, 1 (2002), 25 – 47. https://doi.org/10.1006/csla.2001.0182

[28] Xiaodong Cui, Liang Gu, Bing Xiang, Wei Zhang, and Yuqing Gao. 2008. Developing high performance ASR in the IBM multilingual speech-to-speech translation system. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*. 5121–5124.

[29] Wayne Xiong et al. 2017. Toward Human Parity in Conversational Speech Recognition. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* 25, 12 (Dec. 2017), 2410–2423.

[30] Dong Yu and Li Deng. 2015. *Automatic Speech Recognition*. Springer-Verlag, London.

[31] Zhengyu Zhou et al. 2006. A multi-pass error detection and correction framework for Mandarin LVCSR. In *INTERSPEECH*.