Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Computing and Information Systems | School of Computing and Information Systems

7-2023

Synthesizing speech test cases with text-to-speech? An empirical study on the false alarms in automated speech recognition testing

Julia Kaiwen LAU

Kelvin Kai Wen KONG

Julian Hao YONG

Per Hoong TAN

Zhou YANG

See next page for additional authors

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the Databases and Information Systems Commons, and the Software Engineering Commons

Citation

LAU, Julia Kaiwen; KONG, Kelvin Kai Wen; YONG, Julian Hao; TAN, Per Hoong; YANG, Zhou; YONG, Zi Qian; LOW, Joshua Chern Wey; CHONG, Chun Yong; LIM, Mei Kuan; and David LO. Synthesizing speech test cases with text-to-speech? An empirical study on the false alarms in automated speech recognition testing. (2023). *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, Seattle, USA, 2023 July 17-21*. 1169-1181.
Available at: https://ink.library.smu.edu.sg/sis_research/8566

This Conference Proceeding Article is brought to you for free and open access by the School of Computing and Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Computing and Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email cherylds@smu.edu.sg.

Author

Julia Kaiwen LAU, Kelvin Kai Wen KONG, Julian Hao YONG, Per Hoong TAN, Zhou YANG, Zi Qian YONG, Joshua Chern Wey LOW, Chun Yong CHONG, Mei Kuan LIM, and David LO

This conference proceeding article is available at Institutional Knowledge at Singapore Management University: https://ink.library.smu.edu.sg/sis_research/8566

Synthesizing Speech Test Cases with Text-to-Speech? An Empirical Study on the False Alarms in Automated Speech Recognition Testing

Julia Kaiwen Lau
jlau0019@student.monash.edu
School of Information Technology, Monash University Malaysia
Subang Jaya, Malaysia

Kelvin Kai Wen Kong
kkon0010@student.monash.edu
School of Information Technology, Monash University Malaysia
Subang Jaya, Malaysia

Julian Hao Yong
jyon0017@student.monash.edu
School of Information Technology, Monash University Malaysia
Subang Jaya, Malaysia

Per Hoong Tan
ptan0021@student.monash.edu
School of Information Technology, Monash University Malaysia
Subang Jaya, Malaysia

Zhou Yang
zyang@smu.edu.sg
School of Computing and Information Systems, Singapore Management University
Singapore, Singapore

Zi Qian Yong
zyon0005@student.monash.edu
School of Information Technology, Monash University Malaysia
Subang Jaya, Malaysia

Joshua Chern Wey Low
clow0007@student.monash.edu
School of Information Technology, Monash University Malaysia
Subang Jaya, Malaysia

Chun Yong Chong
chong.chunyong@monash.edu
School of Information Technology, Monash University Malaysia
Subang Jaya, Malaysia

Mei Kuan Lim
lim.meikuan@monash.edu
School of Information Technology, Monash University Malaysia
Subang Jaya, Malaysia

David Lo
davidlo@smu.edu.sg
School of Computing and Information Systems, Singapore Management University
Singapore, Singapore

arXiv:2305.17445v3 [cs.SE] 19 Jul 2023

ABSTRACT

Recent studies have proposed the use of Text-To-Speech (TTS) systems to automatically synthesise speech test cases on a scale and uncover a large number of failures in ASR systems. However, the failures uncovered by synthetic test cases may not reflect the actual performance of an ASR system when it transcribes human audio, which we refer to as *false alarms*. Given a failed test case synthesised from TTS systems, which consists of TTS-generated audio and the corresponding ground truth text, we feed the human audio stating the same text to an ASR system. If human audio can be correctly transcribed, an instance of a *false alarm* is detected.

In this study, we investigate false alarm occurrences in five popular ASR systems using synthetic audio generated from four TTS systems and human audio obtained from two commonly used datasets. Our results show that the least number of false alarms is identified when testing Deepspeech, and the number of false alarms is the highest when testing Wav2vec2. On average, false alarm rates range from 21% to 34% in all five ASR systems. Among the TTS systems used, Google TTS produces the least number of false alarms (17%), and Espeak TTS produces the highest number of false alarms (32%) among the four TTS systems. Additionally, we build a false alarm estimator that flags potential false alarms, which achieves promising results: a precision of 98.3%, a recall of 96.4%, an accuracy of 98.5%, and an F1 score of 97.3%. Our study provides insight into the appropriate selection of TTS systems to generate high-quality speech to test ASR systems. Additionally, a false alarm estimator can be a way to minimise the impact of false alarms and help developers choose suitable test inputs when evaluating ASR systems. The source code used in this paper is publicly available on GitHub at https://github.com/julianyonghao/FAinASRtest.

‡Zhou Yang is the corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2023 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Automated Speech Recognition, Software Testing, False Alarms

ACM Reference Format:

Julia Kaiwen Lau, Kelvin Kai Wen Kong, Julian Hao Yong, Per Hoong Tan, Zhou Yang, Zi Qian Yong, Joshua Chern Wey Low, Chun Yong Chong, Mei Kuan Lim, and David Lo. 2023. Synthesizing Speech Test Cases with Text-to-Speech? An Empirical Study on the False Alarms in Automated Speech Recognition Testing. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Automatic Speech Recognition (ASR) allows users to interact with devices using their voices. ASR systems are prevalent in our daily lives [51]. Popular applications of ASR systems include mobile virtual assistants such as Siri[1] and Alexa.[2] As ASR systems become increasingly popular, it is crucial to ensure that ASR systems are capable of correctly recognising speech. To test an ASR system, an audio input and its corresponding transcription are required, which we call a *speech test case*. Collecting speech test cases manually, where recording and transcribing audio inputs are largely human-based, is rather laborious and costly. As a result, it is critical to automate the process of generating speech test cases on a scale [24].

Recent studies propose automated testing methods for ASR systems by using Text-To-Speech (TTS) systems, such as CrossASR [3]. TTS is a technology that accepts text as input and produces audio as output [31]. Advances in deep neural networks have made TTS systems more powerful, accessible, and inexpensive to use. CrossASR uses TTS systems to automatically generate test cases for ASR systems. Instead of manually recording and transcribing audio, a text (ground truth) is fed into the TTS to produce TTS-generated audio, which in return is used to test the ASR system. Without the need to manually record and transcribe audio, ASR system developers can quickly scale up and automate the testing process by feeding a large text corpus into the TTS to produce test cases.

However, ASR systems are mainly trained and tested to recognise human audio in practical environments. One question of using TTS-generated audio to test ASR systems naturally arises: *Do failed test cases uncovered by TTS-generated audio reveal real faults when the ASR system transcribes human audio?* An ASR system may fail to transcribe TTS-generated audio, but can correctly transcribe human audio stating the same text. Such a synthetic failed test case does not suggest an issue with the ASR as it is able to correctly recognise human audio. Instead, it suggests an issue with test case generation using TTS, which we refer to as *false alarms*.

False alarms can have a significant impact on the evaluation and improvement of ASR systems. During the testing process of a software system, it is usually indicative of a code defect when a test case fails. However, not all test failures imply that there are code defects; the failure may be due to faulty test cases [18]. Similar cases can occur when testing ASR systems with TTS-generated audio. When false alarms occur, developers need to troubleshoot the ASR systems to identify the issue, not knowing that the issue is actually caused by the test case using TTS-generated audio. Such false alarms provide less information on the improvement of ASR systems and impede the testing process. Although some tools have been proposed to identify false alarms in conventional software testing [16, 26, 28, 50], there is still a gap in false alarm identification for ASR testing as many of these tools target false alarms caused by static analysis tools, which are used to identify potential bugs and security vulnerabilities in source codes.

There are a variety of factors that can cause false alarms. One of the factors is that, since the main function of an ASR system is to transcribe human audio in operational environments, many state-of-the-art ASR systems are trained with human audio rather than TTS-generated audio. For example, Deepspeech [17] is trained with 5,000 hours of audio by 9,600 speakers. Deepspeech2 [2] is also trained with large amounts of labelled data consisting of human audio with the corresponding transcriptions. Although modern TTS systems try to mimic human speech, they can potentially lack certain aspects of human speech, e.g., variations in pitch, intensity, duration, and speech sections [40]. Additionally, TTS-generated audio generally lacks natural phonetic variability [36] and lacks redundant acoustic cues present in natural speech [43]. The above factors make the TTS-generated audio follow a data distribution that is different from that of an ASR system's training data. Deep neural network (DNN)-based systems are known to perform poorly in out-of-distribution data. Thus, the failure identified using TTS-generated audio may not reflect the actual performance of an ASR system.

The primary objective of this research is to perform an empirical assessment of false alarms in automated speech recognition (ASR) testing and develop a false alarm estimator. The estimator serves as a tool to anticipate potential false alarms. To achieve this, we conducted experiments using five distinct ASR systems and four TTS systems. The ASR systems are Deepspeech [17], Deepspeech2 [2], Vosk [8], Wav2letter++ [9], and Wav2vec2 [7]. The TTS systems include Google TTS [11], Espeak [13], Festival [41],and GlowTTS [29]. We use the LJ Speech Dataset [22] and the LibriSpeech Dataset [32], which consist of 9,925 and 12,000 pairs of text and human audio, respectively.

Of all test cases synthesised using TTS systems, 20.79%, 32.25%, 12.71%, 25.82%, and 34.16% of failed test cases are identified as false alarms when tested with Deepspeech, Deepspeech2, Vosk, Wav2letter, and Wav2vec2, respectively. While Vosk has produced the least number of false alarms, its performance differs when tested on different datasets. On the LJ Speech Dataset, Vosk achieves a low Word Error Rate (WER) value of 0.1 when transcribing human audio. However, Vosk performs poorly on the human audio from the LibriSpeech Dataset, which indicates that the ASR performs in an inconsistent manner. DeepSpeech, on the other hand, performs consistently on both LJ Speech and LibriSpeech Datasets and yields a reasonably low number of false alarms. The highest number of false alarms is identified with the use of Wav2vec2.

Among all the Text-To-Speech (TTS) systems assessed, it is observed that the ASR systems that use Google TTS and GlowTTS exhibit the fewest instances of false alarms. This finding suggests that both TTS systems are proficient at generating superior quality speech test cases. When manually reviewing false alarms, we find that a substantial number of them are attributed to the generation of unclear speech audio by the TTS systems. Moreover, the false alarm estimator we propose, based on Recurrent Neural Networks

[1] https://www.apple.com/siri/
[2] https://developer.amazon.com/en-GB/alexa

(RNN), is capable of estimating the potential occurrences of false alarms.

The contributions of our paper are as follows:

- We have evaluated the performance difference in ASR systems when transcribing TTS-generated audio and human audio and showed that ASR systems are generally better in transcribing human audio.
- We identify a large number of false alarms when testing ASR systems with TTS-generated audio, raising awareness to developers who use TTS systems to test ASR systems.
- Among all the TTS systems used, the ASR systems can effectively transcribe audio generated using Google TTS and GlowTTS. Developers can consider selecting Google TTS and GlowTTS to generate high-quality speech test cases to test ASR systems.
- We propose a false alarm estimator that flags possible false alarms when ASR is tested with TTS-generated audio.

Paper Structure: Section 2 provides an overview of ASR systems, ASR testing, and CrossASR. In Section 3, we present our approach to identify false alarms and train a false alarm estimator. Section 4 outlines our datasets and experimental settings. Our research questions and results are addressed in Section 5, followed by a discussion of the findings in Section 6. Additionally, we explore related work in Section 7, address threats to validity in Section 8, and conclude our work and discuss future research in Section 9.

2 BACKGROUND

2.1 Automated Speech Recognition Systems

Automated Speech Recognition (ASR) systems convert audio (input) into text to obtain textual information on the given speech [25, 51]. The underlying architecture of an ASR primarily involves the process of accepting an input speech sequence $S = \{s_1, s_2, ..., s_N\}$ and recognising them as textual token sequence $T = \{t_1, t_2, ..., t_M\}$, which usually is in the form of phonemes, grapheme or word pieces [25]. The following equation demonstrates the goal of an ASR system, which is to find the best textual token sequence $\hat{T}$. This is done by selecting a T from a collection of all tokens, V, with the highest probability given a speech sequence S.

$$\hat{T} = \underset{t \in V}{arg\,max}\, p(T \mid S) \tag{1}$$

2.2 ASR Testing

Researchers have proposed different methods for ASR testing, such as CrossASR [3], ASRTest [25], and PROPHET [46]. CrossASR [3] uses differential testing and does not require manual labelling of data. CrossASR uses TTS systems to automatically generate test cases for ASR systems to efficiently build test cases to uncover erroneous behaviour in ASR systems. ASRTest [25] is an automated testing approach built on metamorphic testing theory. The primary objective of ASRTest is to enhance the robustness of ASR systems. It emphasises the importance of ASR systems capable of accurately transcribing speech audio under diverse conditions, encompassing variations in characteristics, background noise, and acoustic distortions originating from the environment. ASRTest achieves this goal by incorporating various speech transformation operators designed to assess the robustness of ASR systems. These operators include the mutation of speech characteristics, noise injection, and simulation of reverberation. Apart from that, PROPHET is a tool that predicts word errors in ASR systems to prioritise test cases that are most likely to be incorrect in order to efficiently uncover more errors for the improvement of ASR systems.

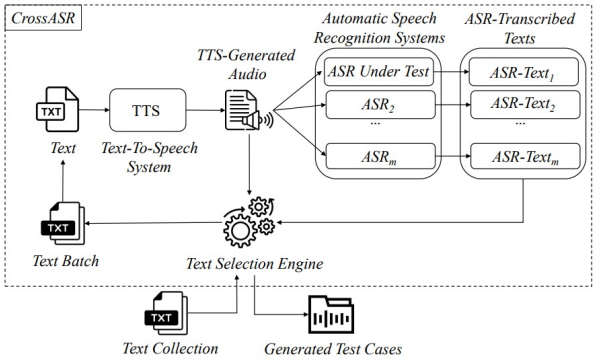

Figure 1: The Architecture of CrossASR [3]. It accepts a corpus of texts (*Text Collection*) as input. Each *Text* is fed into the Text-To-Speech System to generate TTS-generated audio. The audio is then processed by ASR system(s), producing *ASR-Transcribed Texts*. These texts are compared with the input *Text* to identify failed test cases.

Given our objective of evaluating false alarms in the context of synthetic test cases, we choose CrossASR [3] as the tool in our experiment. CrossASR's ability to generate test cases using TTS systems aligns well with our research focus, enabling us to thoroughly assess and analyse false alarms in ASR systems testing.

2.3 CrossASR

CrossASR serves the purpose of automating the generation of test cases for ASR systems without the need for manually labelled data. This is achieved by using Text-to-Speech (TTS) systems within the CrossASR framework. By providing a text input, CrossASR utilises TTS systems to synthesise speech, creating TTS-generated audio for testing. One of the key techniques employed by CrossASR is differential testing. This approach involves comparing and contrasting the outputs of multiple ASR systems when presented with the same input. By analysing and identifying inconsistencies or incorrect behaviours among the outputs of the ASR system, CrossASR effectively detects and highlights areas where ASR systems may be functioning improperly.

Figure 1 shows the architecture of CrossASR. A batch of texts called *Text Batch* is first selected from the *text collection* via the *test selection engine*. The *text batch* contains a number of *texts*. Then, each *text* in *text batch* is fed into *Text-To-Speech system*, which produces a TTS-generated audio of the text. The *TTS-generated audio* is then fed into the ASR systems. ASR systems then transcribe *TTS-generated audio*, producing *ASR-transcribed texts*. The *ASR-transcribed texts* are then compared with the input *text* to identify failed test cases. If the *ASR-transcribed text* produced by the ASR under test matches the *text*, CrossASR determines it as a *successful* test case. If the *ASR-transcribed text* produced by the ASR under test

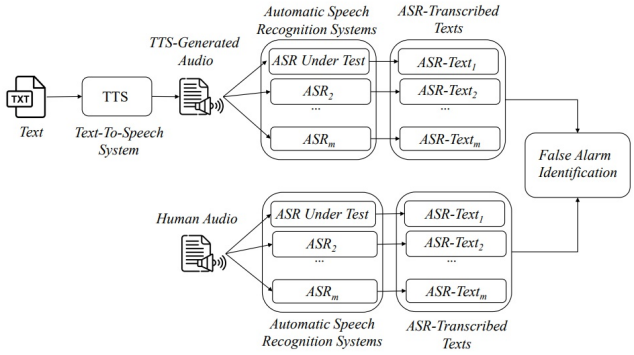

Figure 2: The overview of our proposed methodology to uncover false alarms.

does not match the input *Text* but there is no other *ASR-transcribed text* that matches the input *Text*, CrossASR considers this a *failed* test case. If all the *ASR-transcribed texts* are incorrect, CrossASR considers this as an *indeterminable* case, as there could be an issue with the TTS.

3 METHODOLOGY

In this section, we present an overview of our preprocessing steps, the methodology used to identify false alarms, and the training process of our false alarm estimator.

3.1 False Alarm Identification

In our research, a *false alarm* refers to a specific situation where an Automatic Speech Recognition (ASR) system exhibits a failure in transcribing audio that is generated using Text-to-Speech (TTS), while it can accurately transcribe the same content when presented with human audio. The inclusion of human audio as a reference is crucial to validate the existence of false alarms when TTS-generated audio is incorrectly transcribed by the ASR system.

Figure 2 shows the overview of our approach to identify false alarms. The input *text* is fed into *Text-To-Speech systems* to produce *TTS-generated audio*. The *TTS-generated audio* is fed into *automatic speech recognition systems* to produce *ASR-transcribed texts*. We then provide *human audio* into *automatic speech recognition engines* to produce another set of *ASR transcribed texts*. Among the ASR systems used, we specifically choose one for testing purposes, known as the "ASR under test" (as indicated in Figure 2). The *ASR-transcribed texts* of human audio and the *TTS-generated audio* will then be compared with the input *Text* to determine whether the test case is a false alarm. A test case is deemed a potential false alarm only if the ASR under test accurately transcribes human audio but incorrectly transcribes TTS-generated audio. If the ASR under test flags the test case as a potential false alarm, we will cross-reference with other ASR systems (that is, ASR_2, ASR_m in Figure 2) to verify the flagged false alarm. It is considered a true false alarm only if the ASR under test and at least one other ASR fail to correctly transcribe TTS-generated audio. In this study, we assume that the false alarm may be due to TTS-generated audio when more than one ASR fails to transcribe the TTS-generated audio.

In our evaluation process, we use the Word Error Rate (WER) to measure the performance of ASR systems. WER is a widely used metric that quantifies the accuracy of ASR by indicating the proportion of incorrectly transcribed words. As described by Kepuška et al. [27], the WER value is proportional to the number of words transcribed erroneously by the ASR system. The WER is determined by calculating the number of word errors, which encompasses the combined sum of insertions (I), deletions (D) and substitutions (S) of words made by the ASR system compared to the reference transcription. This sum is then divided by the total number of words in the reference transcription (N), as represented by Equation (2).

$$WER = \frac{I + D + S}{N} \tag{2}$$

A higher WER value indicates a larger number of incorrectly transcribed words by the ASR system, reflecting a decrease in overall transcription accuracy. By employing the WER metric, we can effectively assess and compare the performance of different ASR systems in terms of their transcription accuracy.

3.2 Text and Audio Processing

Text Processing. Prior to being input into the Text-to-Speech (TTS) system, a *text* is subjected to a preprocessing stage. The objective of this step is to facilitate accurate pronunciation by standardising the text format. Similarly, text processing is performed on *ASR-transcribed texts* before *false alarm identification* to standardise the transcriptions. This standardisation allows for accurate comparisons when identifying false alarms. Since different ASR systems may transcribe audio in varying ways, slight formatting differences can occur between the *text* and the *ASR-transcribed texts*. These discrepancies should not be misconstrued as instances of false alarms, as the intrinsic content remains consistent. For instance, an ASR system might interpret the spoken word "one" as the numerical representation "1," but such a difference in form should not be classified as a false alarm, as the inherent meaning is the same. Specifically, the text processing steps are as follows:

- Converting all characters to lowercase.
- Remove all types of punctuation, except apostrophes.
- Resolving abbreviations by expanding them to their full form (e.g., “Mr” to “Mister”).
- Changing numerals to words (e.g. "1" to "one").

Audio Processing. Before the audio is fed into the ASR systems for transcription, it undergoes a transformation process. The original audio file is converted to a `.wav` format, using a sampling rate of 16 kHz and a bit depth of 8. This configuration is widely acknowledged as the optimal setting for ensuring maximum speech intelligibility throughout the transcription process [39].

3.3 False Alarm Estimator

The motivation behind developing a false alarm estimator can be understood through the following scenarios. First, the existence of false alarms in the ASR performance evaluation can lead testers to draw incorrect conclusions about the actual performance of the system. It also highlights the inefficiency in the testing process, as the resources allocated to identifying these false alarms could have been better used to uncover genuine errors. Second, in the

development life cycle of an ASR system, failed test cases are analysed and used to improve the ASR system (e.g., by fine-tuning or retraining). However, false alarms do not produce further improvements. Therefore, integrating an estimator in this scenario allows users to identify potential occurrences of false alarm from the test inputs, help increase the testing efficiency, and facilitate better data collection to improve ASR systems.

Next, we discuss the components of the proposed false alarm estimator. We use the false alarm results produced from our proposed approach presented in Figure 2 as the training dataset and leverage a supervised learning approach to train a text-based false alarm estimator model. The false alarm estimator aims to estimate whether a false alarm will arise when a word or sentence is provided as input. There are two main reasons for our decision to train a text-based estimator instead of an audio-based one.

One consideration is the simplicity of processing the data. Audio data are typically represented as continuous, time-varying signals, with multiple data points, multiple channels, and variations of amplitude. It is much more complex than text data, which can be represented as a sequence of words. Apart from that, processing audio often requires complex signal processing such as Fourier analysis or spectral analysis. Text data, in comparison, can be processed using simpler techniques such as tokenization or word embedding.

Another consideration is efficiency. The multidimensional nature and complex temporal structure of audio data require larger and more sophisticated classifiers. As a result, audio-based classifiers usually take longer to train and infer [38]. The text-based model proposed in our study can demonstrate better efficiency, allowing us to apply the estimator to large-scale settings.

In this experiment, we used 20 sets of false alarm results obtained from conducting our experiments with four TTS systems (Google, Festival, Espeak, and GlowTTS) and five ASR systems (DeepSpeech, DeepSpeech2, Vosk, Wav2letter++, and Wav2vec2). Each text in a dataset is used to generate 20 synthetic test cases, which can potentially be false alarms. We count the occurrences of false alarms of each text and rank them according to their total number of occurrences. We use the middle number (i.e., 10) as the threshold to prepare examples to train and evaluate our estimator. More specifically, if a text yields more than 10 false alarms, it is assigned label 1. Otherwise, a text has the label 0. The label quantifies the probability that a text leads to more false alarms when used to generate TTS-generated audio for test ASR systems. The words in each text in the resulting labelled dataset are then used to build a vocabulary dictionary using the TensorFlow Tokenizer [1]. The tokenizer assigns an index to each word present in the dataset. The vocabulary dictionary is sorted by frequency of words. Words that have a lower index mean that they appear more frequently in the dataset. An example is shown below:

An example of vocabulary dictionary

[("the", 1), ("of", 2), ("and", 3), ("to", 4), ("in", 5), ("a", 6), ("was", 7), ("that", 8), ("he", 9), ("his", 10)] The word "the" has the lowest index, indicating that it appears the most in the dataset.

With the vocabulary dictionary ready, each sentence in the dataset is assigned to its numerical form, and each integer represents the index of the word in the vocabulary dictionary. Zeros are padded at the end of each sentence to make all numerical sentence representations of equal length. The following shows an example where each word in the sentence is mapped to its index: "the" is mapped to 1 and "green" is mapped to 614. The dataset with all the numerical representations of the sentences along with their labels is divided into 60% for model training and 30% for model testing. 10% of the training set is treated as the training validation set.

An example of output

Input: the green plant owes its power to absorb the energy of sunlight
Mapped output: [1 614 615 7293 55 369 2 4279 1 673 2 2219 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

A Recurrent Neural Network (RNN) model is built with two layers of Long-Short-Term Memory Units (LSTM) [19]. Binary cross-entropy is set as the loss function, and Adam[3] as the optimiser for model training. We train the model for 20 epochs. Finally, the testing dataset is used as unseen data to evaluate model performance.

4 EXPERIMENTS

4.1 Datasets

The datasets used in this paper are open-source speech datasets, namely the LJ Speech Dataset [22] and the LibriSpeech Dataset [32]. These datasets are selected due to their wide usage and popularity in numerous previous studies [30, 33, 34]. LJ Speech Dataset is a collection of short audio clips of a single speaker reading passages from a number of books. It comprises a total of 13,100 English audio recordings, with each recording being no longer than 10 seconds. In total, the dataset provides approximately 24 hours of audio content. Each of these recordings, which were recorded from 2016 to 2017 by the LibriVox[4] project, is accompanied by their corresponding transcriptions. After dropping the unusable recordings (e.g. audio without transcription, unclear audio, mismatch between the audio and the corresponding transcription), the LJ Speech Dataset is reduced to 9,925 recordings that are deemed suitable for use in our research. LibriSpeech Dataset is part of the LibriVox project consisting of approximately 1,000 hours of audiobook recordings from Project Gutenberg,[5] a library of free eBooks. Due to hardware and constraints related to computing resources, we have randomly chosen 12,000 recordings with a total duration of approximately 50 hours from the LibriSpeech Dataset for this research.

4.2 ASR and TTS Systems

The ASR systems chosen for this study are: Deepspeech [17], Deepspeech2 [2], Vosk [8], Wav2letter++ [9], and Wav2vec2 [7]. Our experiment uses these four ASRs because they are popular and widely used in many studies [2, 7, 9, 17]. Deepspeech2, developed by Baidu, is widely recognised and used in various related products from Baidu [7, 25]. Additionally, we select Vosk to represent

[3]Adam is an optimiser algorithm [53] based on stochastic gradient descent that aims to adjust the learning rate to different parameters.
[4]https://librivox.org/
[5]http://www.gutenberg.org

lightweight models due to its widespread adoption in portable devices. Vosk offers speech recognition capabilities for more than 20 languages and has 5.2k stars on GitHub.

The investigated TTS systems are Google TTS [11], Espeak [13], Festival [41], and GlowTTS [29]. In our experiment, we include the first three TTS systems because CrossASR uses them in their study. Additionally, GlowTTS is also introduced into our study to act as a control because it was not originally included in CrossASR.

5 RESULTS

RQ1: How notable is the performance difference between ASR systems when transcribing human audio and TTS-generated audio?

In this research question, our aim is to evaluate the performance variation in each ASR when transcribing TTS-generated and human audio. To achieve this, we run our approach on both the LJ Speech Dataset and the LibriSpeech Dataset. We apply our approach to all possible combinations of ASR and TTS systems. Following this, we calculate the average Word Error Rate (WER) to quantify the discrepancies in performance across different scenarios.

The average WER for each ASR generated using human audio and TTS-generated audio for each combination of TTS and ASR systems is shown in Figure 3. The graph is divided into two sections: the top section represents results from the LJ Speech Dataset, while the bottom section corresponds to the LibriSpeech Dataset. The vertical axis denotes the average WER and the horizontal axis indicates the different ASR systems. In each section, there are five graphs where every graph corresponds to a different audio type: human audio and audio generated by the different TTS systems.

Overall, in both datasets, ASR systems have delivered lower WER values when transcribing human audio than TTS-generated audio. As shown in Figure 3, most ASR systems produce a lower WER when transcribing human audio. On the contrary, these ASR systems exhibit higher WER values when working with TTS-generated audio. This implies that ASR systems generally are better at transcribing human audio than TTS-generated audio. On average, ASR systems produce an average WER value of 0.27 when transcribing TTS-generated audio and an average WER value of 0.10 with human audio. There are some cases where an ASR is able to achieve similar low WER results with both TTS-generated audio and human audio. One such example is Wav2Vec2, where it has yielded a WER value of 0.08 and 0.078 on Google-generated audio and human audio, respectively, in the LJ Speech Dataset.

However, there is an exception to this common pattern. Unlike the result in the LJ Speech Dataset shown in Figure 3, Vosk performs better on TTS-generated audio than human audio in the LibriSpeech Dataset. This anomaly may arise from the characteristics of Vosk as a lightweight model. It suggests a possible compromise between the model's accuracy and its adaptability for use on lightweight devices, which might be the reason for Vosk's lower accuracy. Even with its relatively lower accuracy, we decided to include Vosk in our study due to its wide adoption in mobile applications.

In addition to that, there is a clear difference in performance between the ASR systems when transcribing audio produced by different TTS systems. ASR systems have produced high average WER values when transcribing the audio produced by the Espeak TTS system. Most ASR systems have generated low WER values when transcribing GlowTTS-generated audio and Google-generated audio. After manually reviewing the audio produced by the TTS systems, we find that both GlowTTS and Google TTS consistently generate the clearest human-like audio. This is based on the manual evaluation by the authors of this work, where we manually sampled 10 audio files generated by each of the TTS systems. Five of the authors evaluated the quality of the sample audio files on a scale of 1 to 5, with 1 being the poorest quality and 5 being the best quality. The average scores given to Espeak, Festival, Google TTS, and Glow TTS are 1.6, 2.2, 4.6, and 4.6 respectively. This shows that Google TTS and GlowTTS tend to produce clear human-like audio files compared to the other TTS systems. As a result, the WER values tend to be lower (better) for both TTS systems. This suggests that the quality of the TTS has an impact on the performance of the ASR.

Answers for RQ1: There are notable differences in the performance of ASR systems when transcribing human audio and TTS-generated audio. For LJ Speech Dataset, the average WER for human and TTS-generated audio are 0.102 and 0.254 respectively, while for LibriSpeech Dataset, the average WER are 0.099 and 0.282 respectively.

RQ2: How prevalent are false alarms when using TTS-generated audio to test ASR systems?

In this research question, we investigate the frequency of false alarms when TTS-generated audio is used to test ASR systems. To do this, we run our proposed pipeline shown in Figure 2 for each combination of ASR and TTS systems and observe the number of test cases that are flagged as false alarms. The results are shown in Figure 4, Table 1 and Table 2. Figure 4 shows a detailed comparison of the percentages of false alarms from test cases for each combination of ASR and TTS systems with the LJ Speech Dataset and the LibriSpeech Dataset. Table 1 and Table 2 show the total number of false alarms generated for all combinations of TTS and ASR systems with the LJ Speech Dataset and LibriSpeech Dataset respectively. The number of false alarms illustrates the limitations of using TTS-generated audio to test ASR systems. The more false alarms are produced, the more unreliable the TTS-generated audio is for testing ASR systems.

In the LJ Speech Dataset, the number of false alarms for Deepspeech, Deepspeech2, Vosk, Wav2letter++ and Wav2vec2 is 4,386, 8,323, 8,351, 7,397 and 9,110 respectively, as shown in Table 1. Deepspeech produces the least number of false alarms with all TTS systems, as shown in Figure 4 whereas Wav2vec2 produces the most false alarms with most TTS systems.

In the LibriSpeech Dataset, the number of false alarms for Deepspeech, Deepspeech2, Vosk, Wav2letter++ and Wav2vec2 are 13,844, 19,961, 2,795, 15,244, and 20,852, respectively, as shown in Table 2. Although Vosk has produced the lowest false alarms with LibriSpeech, it cannot be taken into account due to the result inconsistency shown in Figure 3. In Figure 3, when tested on the LibriSpeech Dataset, Vosk is shown to have produced a higher WER when transcribing human audio than the audio generated by some of the TTS systems. This means that Vosk is unable to transcribe human audio in LibriSpeech Dataset, and there are a lot of failed test cases for human audio. With Vosk struggling to transcribe human audio from

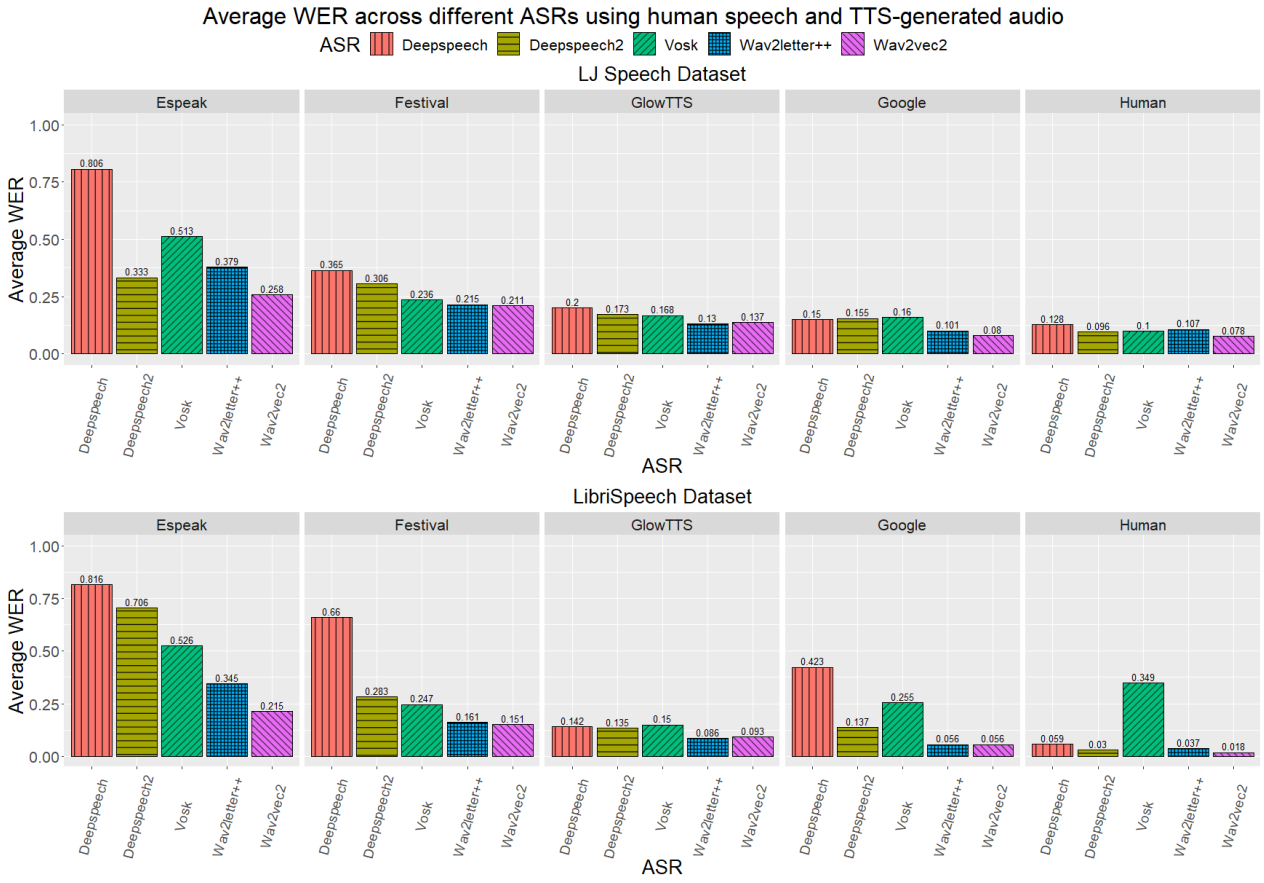

Figure 3: Average WER for each combination of TTS and ASR systems with human audio and TTS-generated audio.

LibriSpeech Dataset, we are unable to validate the occurrences of false alarms with Vosk and thus Vosk is omitted for analysis. With Vosk excluded, we can observe that Deepspeech produces the least number of false alarms, and Wav2vec2 yields the most false alarms, which is similar to the results from the LJ Speech Dataset. Thus, we can consider Deepspeech as the best ASR in terms of false alarm occurrences. Due to the large amounts of test cases required to test a software system, even a small percentage of false alarms will cost developers a lot of time and effort to resolve. Developers have the expectation that a testing tool should produce fewer than 10% false alarms to avoid fixing non-existent issues [37], indicating that 10% should be the threshold of acceptable false alarm rate. Therefore, looking at our results in Figure 4, many of the false alarm rates have exceeded 10%, which means that false alarms are prevalent when using TTS-generated audio to test state-of-the-art ASR systems.

In Research Question 1 and in Figure 3, we note that the quality of TTS affects the performance of ASR, with Google TTS and GlowTTS being the best quality among the other TTS systems. In Figure 4, we also observe that there is a correlation between the quality of TTS systems and the number of false alarms. ASR systems have generated the lowest false alarms with Google TTS and the second lowest with GlowTTS. In contrast, all chosen ASR systems produce the most false alarms when transcribing Espeak-generated audio. This is consistent with the WER results in Figure 3, and this observation suggests that the quality of the TTS-generated audio

Table 1: LJ Speech - Number of False Alarms

	DS	DS2	VK	W2L	W2V2	Total
Google	627	1,223	1,211	703	850	**4,614**
Festival	1,274	2,368	2,309	2,235	2,960	**11,146**
Espeak	1,469	2,897	3,048	2,906	3,000	**13,320**
GlowTTS	1,016	1,835	1,783	1,553	2,300	**8,487**
Total	4,386	8,323	8,351	7,397	9,110	**37,567**

DS: Deepspeech, DS2: Deepspeech2, VK: Vosk, W2L: Wav2letter++, W2V2: Wav2vec2

Table 2: LibriSpeech - Number of False Alarms

	DS	DS2	VK	W2L	W2V2	Total
Google	2,998	4,110	541	2,617	3,470	**13,736**
Festival	3,707	5,377	725	4,325	5,900	**20,034**
Espeak	3,805	5,631	851	4,694	6,236	**21,217**
GlowTTS	3,334	4,843	678	3,608	5,246	**17,709**
Total	13,844	19,961	2,795	15,244	20,852	**72,696**

DS: Deepspeech, DS2: Deepspeech2, VK: Vosk, W2L: Wav2letter++, W2V2: Wav2vec2

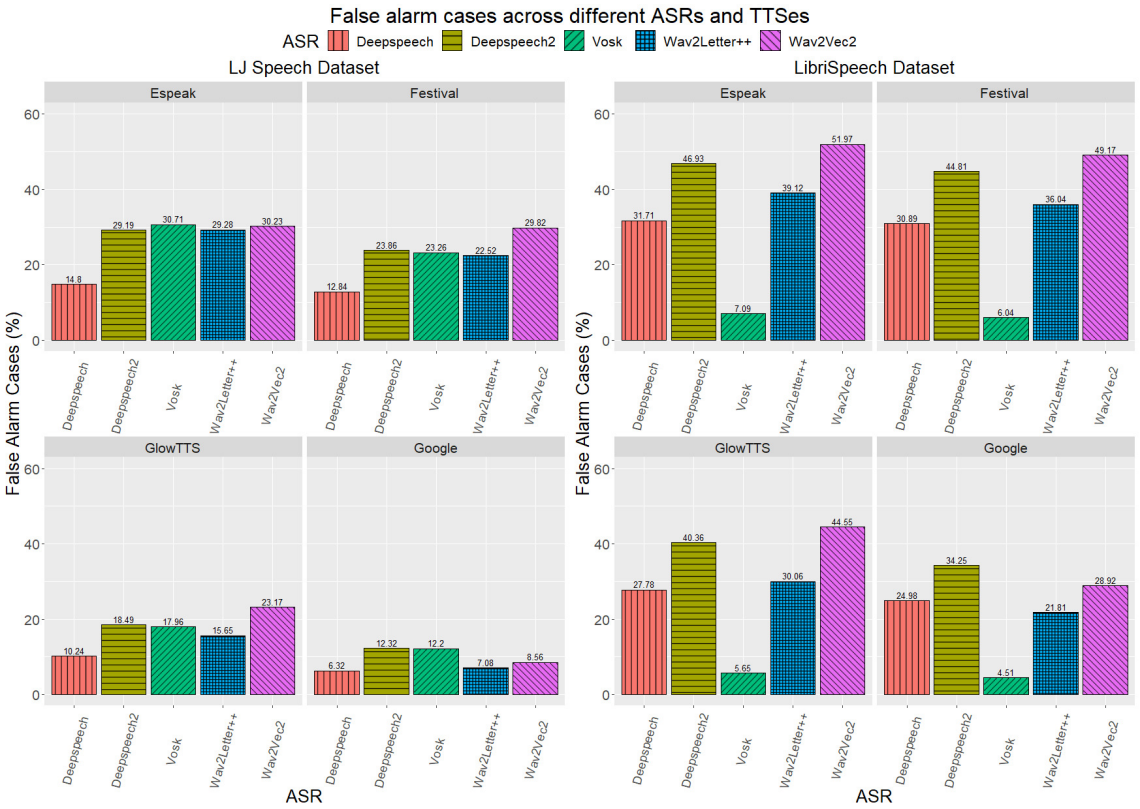

Figure 4: Percentage of false alarms out of all the test cases for each combination of TTS and ASR systems

plays a crucial role in the occurrence of false alarms. Therefore, using a higher quality TTS will result in a lower WER value, and thus fewer false alarms. The lower the percentage of false alarm occurrence, the more preferable the TTS is for automated testing of ASR systems.

Answers for RQ2: The occurrence of false alarms is prevalent when using TTS-generated audio to test ASR systems. When evaluated on LJ Speech Dataset, only experiments conducted using Google TTS are found with fewer than 10% false alarms. For all other ASR and TTS systems, on average false alarm cases are found to exceed 15%. The results also suggest that the prevalence of false alarms depends on the audio quality of TTS-generated audio.

RQ3: How to estimate the potential occurrences of false alarms when testing ASR systems?

We use the results obtained in RQ2 to train a Recurrent Neural Network (RNN) classifier to estimate whether a test case is likely to lead to false alarms. Training and test data are pre-processed as discussed in Section 3.3 to transform the results into a suitable input format for the RNN. The dataset containing 4,769 positive and 17,156 negative examples, is randomly shuffled and split into 60% for model training, 10% as a validation dataset, and 30% for model testing, resulting in 13,156, 2,192 and 6,577 cases, respectively. The model is trained with 20 epochs. Binary cross entropy is used as a loss function for model fitting.

We use precision, recall, accuracy, and the F1 score for model performance evaluation. On the unseen test set, the model achieves a high precision of 98.3%. This indicates that 98.3% of the estimated positive instances are actual false alarms, showing that there is a low false positive rate. The recall is 0.964, indicating that our model correctly estimates 96. 4% of the actual false alarms in the test data set. The F1 score for our model is 97.3%, demonstrating its ability to estimate true positives (false alarms).

By observing the results estimated by the false alarm estimator, we aim to gain insight into the specific words or linguistic features that could be contributing to the occurrence of false alarms. These words exhibited distinctive properties that can challenge TTS systems in synthesising clearly and misleading ASR systems, resulting in false alarms. A prime example of such words are homophones. Further elaboration on this finding can be found in Section 6.1.

Answers for RQ3: The evaluation results show that our proposed RNN model is capable of automatically estimating potential false alarms. Our model has shown a precision of 98.3%, a recall of 96.4%, an accuracy of 98.5%, and an F1 score of 97.3%. The ability to estimate potential occurrences of false alarms can be leveraged as a complementary mechanism to carefully choose test inputs for testing ASR systems.

6 DISCUSSION

6.1 Manual Analysis of False Alarms

We present examples of false alarms and their patterns observed by manual analysis. The analysis procedure is as follows:

Data filtering. For results of each ASR-TTS combination, we compile a list of words that ASR systems incorrectly transcribe when processing TTS-generated audio. Then, we focus on the top 10 words that are most frequently incorrectly transcribed. From there, we select false alarms where at least half of the cross-reference ASR systems have flagged those cases as false alarms. This is done to ensure that the cases that we analyse affect the majority of ASR systems, and we can compare the incorrect transcriptions to observe a pattern. Given time constraints and labour intensiveness of manual inspection, we analyse a sample size of 500 false alarms, which is larger than the statistically representative sample size computed with a confidence level of 95% and a confidence interval of 5.

Pattern finding. To ensure consistency and reliability in our findings while observing patterns, the manual analysis is conducted by five authors of this paper in the following manner.

(1) The audio files of the chosen false alarm cases (TTS-generated audio) are listened to manually.
(2) Each author compares the false alarm audio files with the ASR-transcribed texts.
(3) Each author compares the false alarm audio files with the ground truth.
(4) Each author compares the patterns they have observed and eliminates infrequent patterns, that is, patterns that occur less than three times.

From these patterns, we can observe certain characteristics of a TTS and limitations in ASR that can result in incorrect transcriptions, and thus leading to false alarms. As such, we have identified the following characteristics and limitations.

6.1.1 TTS System's Pronunciation. In our analysis, we frequently find that pronunciation is a major contributing factor to incorrect transcriptions by ASR systems. The following are the most common pronunciation problems:

TTS cannot pronounce consonants prominently

One of the common pronunciation problems observed in TTS-generated audio is the vague pronunciation of consonants, which often leads to inconsistent transcriptions by ASR systems. For example, some TTS systems fail to pronounce the word "r" prominently. An example of this is "officers". When the "r" in "officers" is not pronounced clearly, it will be transcribed as "offices" by the ASR systems.

Original Text:
committee of cabinet **officers** as our government has become more complex

ASR Transcribed Text using TTS-generated audio:
committee of cabinet **offices** as our government has become more complex

TTS cannot pronounce suffixes clearly

Another issue with TTS is their inability to clearly pronounce suffixes, leading to incorrect transcriptions. Some common examples of suffixes that are mistranscribed by ASR systems are "-ing" and "-ed"/"-d". For example, the "-ed" in "asked" is not pronounced prominently by the TTS, and hence the ASR transcribes it as "ask".

Original Text:
we **asked** the nation to turn over all its privately held gold dollars for dollars to the government of the united states

ASR Transcribed Text using TTS-generated audio:
we **ask** the nation to turn over all its privately held gold dollars for dollar to the government of the United States

TTS cannot pronounce grammatical words

Grammatical words consist of words such as articles, conjunctions, and pronouns. Several of them are spelt and pronounced quite similarly to each other. If a TTS fails to pronounce these words clearly, the words in the ground truth texts can be mistaken for other grammatical words by ASR systems. An example of this is the word "as"; if the "a" in "as" is not pronounced prominently, it can lead to the word sounding like "is".

Original Text:
she identified lee harvey oswald **as** the man who shot the policeman

ASR Transcribed Text using TTS-generated audio:
she identified lee havey oswald **is** the man who shot the policeman

6.1.2 ASR Systems Fail to Transcribe Homophones. Homophones are words that are pronounced similarly but have different definitions and spellings. To illustrate, "brake" and "break" are homophones. This is still an open challenge in ASR systems that have difficulty differentiating between homophones. For instance, "toll" and "tole" are homophones used in the example shown below, where both pronunciations sound similar but the ASR is unable to determine the ground truth. Therefore, further investigation is needed to improve the ability of ASR systems to transcribing homophones.

Original Text:
and no doubt made the meat also pay **toll**

ASR Transcribed Text using TTS-generated audio:
and no doubt made the meat also pay **tole**

6.1.3 ASR Systems Fail to Transcribe Words with Multiple Pronunciations. Some words may have different pronunciations depending on the speaker's accent. This raises an issue where the ASR cannot transcribe the word appropriately due to the difference in pronunciation. As an example, the name "marley" is pronounced "mar-lei" in human audio, whereas in the TTS it pronounces "Marley" as "mar-li", leading the ASR to transcribe it as "marly". Therefore, further improvements can be made to the ASR's capability to handle words with multiple pronunciations.

Original Text:
robert marley at the time of his arrest called himself a surgical instrument maker

ASR Transcribed Text using TTS-generated audio:
robert marly at the time of his arrest called himself a surgical instrument maker

7 RELATED WORK

7.1 Testing ASR Systems

Researchers have proposed a series of works to evaluate different properties (e.g. robustness [14, 44], ethics [6, 42], security [48]) of various AI systems (e.g., code models [47, 49], reinforcement learning [15], image recognition [45]). Researchers have also recently proposed various methods to evaluate the quality of ASR systems from various aspects. In this section, we present previous studies on the testing of ASR systems.

Iwama and Fukuda [23] evaluate the basic recognition capability of ASR systems. They use a language model to generate test sentences and use an audio converter (i.e. TTS systems) to generate various audio data. However, the audio generated using TTS systems may be invalid. To address this concern, Asyrofi et al. [3] propose the use of differential testing (CrossASR) to filter potentially invalid failed test cases by cross-referencing the output of different ASR systems. The intuition is that if none of the ASR systems can correctly recognise TTS-generated audio, it may be due to the reason that this TTS-generated audio itself is of low quality. On top of CrossASR [3], Asyrofi et al. further propose CrossASR++ [4], which incorporates more ASR and TTS systems. Yuen et al. [52] apply various text transformations to generate more failed test cases. Recent studies also show that synthesised speech data can also be used to improve the performance of ASR systems [5, 12, 21, 54]. To better improve find the valuable test cases that can improve ASR systems, Yang et al. [46] design Prophet to prioritise speech test cases using a BERT-based language model.

There is also a line of work on applying transformations to audio to evaluate the robustness of ASR systems. Du et al. [10] propose DeepCruiser, whose main objective is to create an automated testing framework for ASR systems based on recurrent neural networks (RNN). DeepCruiser can be applied to RNN-based ASR systems, while it is not applicable to ASR systems that use the latest transformer-based architecture (e.g., HuBERT [20]). DeepCruiser incorporates eight metamorphic transformations, such as audio speed variation, into the original audio input to generate new audio test inputs. Wu and Rajan [44] use frequency masking to transform audio to change the output of ASR systems (i.e., robustness evaluation). Ji et al. [25] propose ASRTest, a tool that uses the metamorphic testing theory. They implement three families of transformation operators that can simulate practical application scenarios to generate speeches. Rajan et al. [35] design aequevox to test the fairness of ASR systems. They found that ASR systems are more robust in the audio spoken by men when noise is added to the audio.

7.2 Detecting False Alarms in Software Systems

Researchers in software engineering have proposed to detect false alarms in various systems. Herzig and Nagappan [18] propose a classification model that uses association mining rules to discover patterns between false alarms. The association rules are then used to estimate and classify failed test cases as false test alarms. The rationale is that false alarms exhibit specific patterns that can be used to identify false alarms. The features used for their model are the unique identifiers of the test case executions, the unique identifiers of the test case, the identifier of the executed test step, and a binary field that indicates whether the test step passed or failed. Another study presented by Yoon et al. [50] proposes a machine learning-based approach to reduce the number of false alarms for automated static analysis tools. Similarly, false alarm patterns are used to train the support vector machine (SVM) model and to classify false alarms using tree-based abstract syntax feature vectors. Although many tools have been proposed to estimate false alarms, those tools mainly focus on software testing, and there is still a gap in the false alarm estimator for ASR testing. In our approach, we have taken inspiration from Herzig and Nagappan"s study [18] where the training features we used are an array or field of values where each element represents an index of a word based on a vocabulary dictionary. The word comes from the transcription of both successful and false alarm test cases. We have also taken inspiration from the study by Yoon, Jin, and Jung [50] to adopt supervised machine learning for our false alarm estimator. One difference is that we chose to use RNN instead of SVM.

8 THREATS TO VALIDITY

Internal Validity. When the ASR under test flags a test case as a false alarm, it may not be a true false alarm, as it can occur due to the limitations of the ASR itself. To mitigate this threat, we cross-reference with other ASR systems to verify the false alarm in the false alarm identification step of our approach. As we have a small number of ASR systems, we consider it a true false alarm only when the ASR under test and another ASR in the pool flag the test case as a false alarm. However, having at least one other ASR failing the test case may not be enough to fully validate the false alarm. Given a scenario in which most cross-referenced ASR systems fail in the test case, we still consider that test case as a false alarm. This would indicate that the false alarm is valid only for the ASR under test and for the minority of cross-referenced ASR systems. Therefore, the false alarm data collected are not fully representative of all ASR systems. Apart from that, for the proposed false alarm estimator, we set a threshold such that if a text yields more than 10 false alarms, it is assigned with the label 1 (flagged as false alarm). This assumption might not hold when the estimator is evaluated with significantly more ASR and TTS. As such, the reported findings for our proposed false alarm estimator can only be confined to the chosen ASR and TTS used in this paper. In the future, especially for research that involves a larger number of ASR systems, this should be mitigated by only considering the false alarm as valid should the majority or at least half of the total cross-referenced ASR systems flag the test case as a false alarm.

External Validity. The results for RQ1 and RQ2 depend on the datasets selected for our experiment. As many of our selected ASR

systems have been trained with the LibriSpeech dataset [2, 7, 9], it can introduce bias where these ASR systems may perform better with said dataset. Despite this, the results observed from these ASR systems do not show consistent high performance. This emphasises the necessity of evaluating these ASRs using the LibriSpeech Dataset to better understand their limitations and strengths. On the other hand, by employing a dataset that contains texts already familiar to the ASR systems, we can more confidently attribute false alarms to synthesised audio from the text (i.e., the quality of the TTS systems). As a step to mitigate bias, we also incorporate the LJSpeech Dataset in our experiments, which is unseen by the ASR systems. The selection of ASR and TTS systems is crucial in our experiment. The TTS quality and ASR's ability in transcribing TTS-generated audio impact the number of false alarms generated in our experiment. To minimise the potential bias that comes from this, we used five ASR systems and four TTS systems.

9 CONCLUSION AND FUTURE WORK

We analyse the difference in performance between ASR systems when transcribing human audio and TTS-generated audio, while also evaluating the prevalence of false alarms in ASR systems tested with TTS-generated audio. We also developed a false alarm estimator to help estimate false alarms. This tool allows developers to efficiently estimate the possible occurrences of false alarms without the need for manual verification of all failed test cases. The trained estimator has a precision of 98.3% and a recall of 96.4% when tested with the LJ Speech Dataset and the LibriSpeech Dataset, showing good signs of possible future adoption for efficient ASR testing. Although TTS-generated audio presents opportunities for cost-effective and time-effective ASR testing, false alarms remain a limitation. Our study highlights the reasons behind false alarms and discusses strategies for addressing this issue, including developing a false alarm estimator to help developers identify false alarms efficiently. We propose to augment CrossASR with the false alarm estimator to identify potential cases of false alarm test cases. Furthermore, the use of high-quality TTS systems can improve the usability of CrossASR and further optimise ASR testing procedures, eliminating the need for manual verification and the creation of test cases.

In the future, we plan to evaluate our false alarm estimator using other datasets. Our false alarm estimator is primarily trained and tested with results obtained from the LJ Speech Dataset and the LibriSpeech Dataset. This suggests that its effectiveness may vary when applied to different datasets. To achieve an estimator model that is applicable across different datasets, we believe cross-dataset testing and further fine-tuning must be carried out. Furthermore, we would like to explore the use of a speech or multimodal approach to train the proposed false alarm estimator to further enhance the performance and capabilities in estimating potential occurrences of false alarms.

In conclusion, developers should be aware of testing with TTS-generated audio, as false alarms have been proven to affect results. This also recommends that researchers consider the active usage of human audio in research, taking into consideration the cons of doing so, such as poor scalability and expense. Alternatively, incorporating a false alarm estimator can still allow researchers to utilise TTS-generated audio by able to find the likely false alarms. With this in mind, we hope that our research presented in this paper can act as a support for the further development of automated ASR testing systems.

ACKNOWLEDGMENT

This research is supported by the Ministry of Education, Singapore under its Academic Research Fund Tier 3 (Award ID: MOET32020-0004). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

REFERENCES

[1] Martín Abadi. 2016. TensorFlow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*. 1–1. https://doi.org/10.1145/2951913.2976746

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. 2016. Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 173–182. https://proceedings.mlr.press/v48/amodei16.html

[3] Muhammad Hilmi Asyrofi, Ferdian Thung, David Lo, and Lingxiao Jiang. 2020. Crossasr: Efficient differential testing of automatic speech recognition via text-to-speech. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 640–650. https://doi.org/10.1109/ICSME46990.2020.00066

[4] Muhammad Hilmi Asyrofi, Zhou Yang, and David Lo. 2021. CrossASR++: A Modular Differential Testing Framework for Automatic Speech Recognition. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1575–1579. https://doi.org/10.1145/3468264.3473124

[5] Muhammad Hilmi Asyrofi, Zhou Yang, Jicke Shi, Chu Wei Quan, and David Lo. 2021. Can differential testing improve automatic speech recognition systems?. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 674–678. https://doi.org/10.1109/ICSME52107.2021.00079

[6] Muhammad Hilmi Asyrofi, Zhou Yang, Imam Nur Bani Yusuf, Hong Jin Kang, Ferdian Thung, and David Lo. 2022. BiasFinder: Metamorphic Test Generation to Uncover Bias for Sentiment Analysis Systems. *IEEE Transactions on Software Engineering* 48, 12 (2022), 5087–5101. https://doi.org/10.1109/TSE.2021.3136169

[7] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems* 33 (2020), 12449–12460.

[8] Alpha Cephei. 2021. Vosk. https://github.com/alphacep/vosk-api Accessed on May 26th, 2023.

[9] Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve. 2016. Wav2letter: an end-to-end convnet-based speech recognition system. *arXiv preprint arXiv:1609.03193* (2016). https://doi.org/10.48550/arXiv.1609.03193

[10] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Jianjun Zhao, and Yang Liu. 2018. Deepcruiser: Automated guided testing for stateful deep learning systems. *arXiv preprint arXiv:1812.05339* (2018). https://doi.org/10.48550/arXiv.1812.05339

[11] Pierre Nicholas Durette. 2020. Google TTS. https://github.com/pndurette/gTTS Accessed on March 18th, 2023.

[12] Amin Fazel, Wei Yang, Yulan Liu, Roberto Barra-Chicote, Yixiong Meng, Roland Maas, and Jasha Droppo. 2021. Synthasr: Unlocking synthetic data for speech recognition. *arXiv preprint arXiv:2106.07803* (2021). https://doi.org/10.48550/arXiv.2106.07803

[13] Source Forge. 2015. eSpeak Text-to-Speech. https://espeak.sourceforge.net/ Accessed on December 25th, 2022.

[14] Xiang Gao, Ripon K. Saha, Mukul R. Prasad, and Abhik Roychoudhury. 2020. Fuzz Testing Based Data Augmentation to Improve Robustness of Deep Neural Networks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1147–1158. https://doi.org/10.1145/3377811.3380415

[15] Chen Gong, Zhou Yang, Yunpeng Bai, Jieke Shi, Arunesh Sinha, Bowen Xu, David Lo, Xinwen Hou, and Guoliang Fan. 2022. Curiosity-Driven and Victim-Aware Adversarial Policies. In *Proceedings of the 38th Annual Computer Security Applications Conference* (Austin, TX, USA) *(ACSAC '22)*. Association for Computing Machinery, New York, NY, USA, 186–200. https://doi.org/10.1145/3564625.3564636

[16] Quinn Hanam, Lin Tan, Reid Holmes, and Patrick Lam. 2014. Finding patterns in static analysis alerts: improving actionable alert ranking. In *Proceedings of the 11th working conference on mining software repositories*. 152–161. https://doi.org/10.1145/2597073.2597100

[17] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. 2014. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567* (2014). https://doi.org/10.48550/arXiv.1412.5567

[18] Kim Herzig and Nachiappan Nagappan. 2015. Empirically detecting false test alarms using association rules. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 39–48. https://doi.org/10.1109/ICSE.2015.133

[19] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1007/978-3-642-24797-2_4

[20] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. 2021. HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29 (2021), 3451–3460. https://doi.org/10.1109/TASLP.2021.3122291

[21] Ting-Yao Hu, Mohammadreza Armandpour, Ashish Shrivastava, Jen-Hao Rick Chang, Hema Koppula, and Oncel Tuzel. 2022. SYNT++: Utilizing Imperfect Synthetic Data to Improve Speech Recognition. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 7682–7686. https://doi.org/10.1109/ICASSP43922.2022.9746217

[22] Keith Ito and Linda Johnson. 2017. The LJ Speech Dataset. https://keithito.com/LJ-Speech-Dataset/.

[23] Futoshi Iwama and Takashi Fukuda. 2019. Automated Testing of Basic Recognition Capability for Speech Recognition Systems. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. 13–24. https://doi.org/10.1109/ICST.2019.00012

[24] Neetu Jain and Rabins Porwal. 2019. Automated test data generation applying heuristic approaches—a survey. In *Software Engineering*. Springer, 699–708. https://doi.org/10.1007/978-981-10-8848-3_68

[25] Pin Ji, Yang Feng, Jia Liu, Zhihong Zhao, and Zhenyu Chen. 2022. ASRTest: Automated Testing for Deep-Neural-Network-Driven Speech Recognition Systems. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual, South Korea) *(ISSTA 2022)*. Association for Computing Machinery, New York, NY, USA, 189–201. https://doi.org/10.1145/3533767.3534391

[26] Hong Jin Kang, Khai Loong Aw, and David Lo. 2022. Detecting false alarms from automatic static analysis tools: how far are we?. In *Proceedings of the 44th International Conference on Software Engineering*. 698–709. https://doi.org/10.1145/3510003.3510214

[27] Veton Këpuska and Gamal Bohouta. 2017. Comparing speech recognition systems (Microsoft API, Google API and CMU Sphinx). *Int. J. Eng. Res. Appl* 7, 03 (2017), 20–24. https://doi.org/10.9790/9622-0703022024

[28] Anant Kharkar, Roshanak Zilouchian Moghaddam, Matthew Jin, Xiaoyu Liu, Xin Shi, Colin Clement, and Neel Sundaresan. 2022. Learning to reduce false positives in analytic bug detectors. In *Proceedings of the 44th International Conference on Software Engineering*. 1307–1316. https://doi.org/10.1145/3510003.3510153

[29] Jaehyeon Kim. 2020. Glow-TTS. https://github.com/jaywalnut310/glow-tts Accessed on May 10th, 2023.

[30] Jason Li, Vitaly Lavrukhin, Boris Ginsburg, Ryan Leary, Oleksii Kuchaiev, Jonathan M Cohen, Huyen Nguyen, and Ravi Teja Gadde. 2019. Jasper: An end-to-end convolutional neural acoustic model. *arXiv preprint arXiv:1904.03288* (2019). https://doi.org/10.48550/arXiv.1904.03288

[31] Joseph P Olive and Mark Y Liberman. 1985. Text to speech—An overview. *The Journal of the Acoustical Society of America* 78, S1 (1985), S6–S6. https://doi.org/10.1121/1.2022951

[32] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 5206–5210. https://doi.org/10.1109/ICASSP.2015.7178964

[33] Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le. 2019. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779* (2019). https://doi.org/10.21437/Interspeech.2019-2680

[34] Daniel S Park, Yu Zhang, Ye Jia, Wei Han, Chung-Cheng Chiu, Bo Li, Yonghui Wu, and Quoc V Le. 2020. Improved noisy student training for automatic speech recognition. *arXiv preprint arXiv:2005.09629* (2020). https://doi.org/10.21437/Interspeech.2020-1470

[35] Sai Sathiesh Rajan, Sakshi Udeshi, and Sudipta Chattopadhyay. 2022. AequeVox: Automated Fairness Testing of Speech Recognition Systems. In *Fundamental Approaches to Software Engineering*, Einar Broch Johnsen and Manuel Wimmer (Eds.). Springer International Publishing, 245–267. https://doi.org/10.1007/978-3-030-99429-7_14

[36] Roy W Roring, Franklin G Hines, and Neil Charness. 2007. Age differences in identifying words in synthetic speech. *Human factors* 49, 1 (2007), 25–31. https://doi.org/10.1518/001872007779598055

[37] Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, and Ciera Jaspan. 2018. Lessons from building static analysis tools at google. *Commun. ACM* 61, 4 (2018), 58–66. https://doi.org/10.1145/3188720

[38] Garima Sharma, Kartikeyan Umapathy, and Sridhar Krishnan. 2020. Trends in audio signal feature extraction methods. *Applied Acoustics* 158 (2020), 107020. https://doi.org/10.1016/j.apacoust.2019.107020

[39] Adriana Stan, Junichi Yamagishi, Simon King, and Matthew Aylett. 2011. The Romanian speech synthesis (RSS) corpus: Building a high quality HMM-based speech synthesis system using a high sampling rate. *Speech Communication* 53, 3 (2011), 442–450. https://doi.org/10.1016/j.specom.2010.12.002

[40] Krista Taake. 2009. A comparison of natural and synthetic speech: with and without simultaneous reading. (2009).

[41] The University of Edinburgh The Center for Speech Technology Research. 2017. The Festival Speech Synthesis System. https://www.cstr.ed.ac.uk/projects/festival/ Accessed on March 18th, 2023.

[42] Zichong Wang, Yang Zhou, Meikang Qiu, Israat Haque, Laura Brown, Yi He, Jianwu Wang, David Lo, and Wenbin Zhang. 2023. Towards Fair Machine Learning Software: Understanding and Addressing Model Bias Through Counterfactual Thinking. https://doi.org/10.48550/ARXIV.2302.08018

[43] Stephen J Winters and David B Pisoni. 2004. Perception and comprehension of synthetic speech. *Research on spoken language processing report* 26 (2004), 95–138.

[44] Xiaoliang Wu and Ajitha Rajan. 2021. Catch Me If You Can: Blackbox Adversarial Attacks on Automatic Speech Recognition using Frequency Masking. https://doi.org/10.48550/ARXIV.2112.01821

[45] Zhou Yang, Jieke Shi, Muhammad Hilmi Asyrofi, and David Lo. 2022. Revisiting Neuron Coverage Metrics and Quality of Deep Neural Networks. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE Computer Society, Los Alamitos, CA, USA, 408–419. https://doi.org/10.1109/SANER53432.2022.00056

[46] Zhou Yang, Jieke Shi, Muhammad Hilmi Asyrofi, Bowen Xu, Xin Zhou, Dong-Gyun Han, and David Lo. 2023. Prioritizing Speech Test Cases. *arXiv preprint arXiv:2302.00330* (2023). https://doi.org/10.48550/arXiv.2302.00330

[47] Zhou Yang, Jieke Shi, Junda He, and David Lo. 2022. Natural Attack for Pre-Trained Models of Code. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1482–1493. https://doi.org/10.1145/3510003.3510146

[48] Zhou Yang, Bowen Xu, Jie M. Zhang, Hong Jin Kang, Jieke Shi, Junda He, and David Lo. 2023. Stealthy Backdoor Attack for Code Models. https://doi.org/10.48550/ARXIV.2301.02496

[49] Noam Yefet, Uri Alon, and Eran Yahav. 2020. Adversarial Examples for Models of Code. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 162 (nov 2020), 30 pages. https://doi.org/10.1145/3428230

[50] Jongwon Yoon, Minsik Jin, and Yungbum Jung. 2014. Reducing false alarms from an industrial-strength static analyzer by SVM. In *2014 21st Asia-Pacific Software Engineering Conference*, Vol. 2. IEEE, 3–6. https://doi.org/10.1109/APSEC.2014.81

[51] Dong Yu and Li Deng. 2016. *Automatic speech recognition*. Vol. 1. Springer. https://doi.org/10.1007/978-1-4471-5779-3

[52] Daniel Hao Xian Yuen, Andrew Yong Chen Pang, Zhou Yang, Chun Yong Chong, Mei Kuan Lim, and David Lo. 2023. ASDF: A Differential Testing Framework for Automatic Speech Recognition Systems. https://doi.org/10.48550/ARXIV.2302.05582

[53] Zijun Zhang. 2018. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. Ieee, 1–2. https://doi.org/10.1109/IWQoS.2018.8624183

[54] Xianrui Zheng, Yulan Liu, Deniz Gunceler, and Daniel Willett. 2021. Using Synthetic Audio to Improve the Recognition of Out-of-Vocabulary Words in End-to-End Asr Systems. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 5674–5678. https://doi.org/10.1109/ICASSP39728.2021.9414778