CrossASR: Efficient Differential Testing of Automatic Speech Recognition via Text-To-Speech

Muhammad Hilmi Asyrofi, Ferdian Thung, David Lo and Lingxiao Jiang
School of Information Systems, Singapore Management University
Email: {mhilmia, ferdianthung, davidlo, lxjiang}@smu.edu.sg

***Abstract*—Automatic speech recognition (ASR) systems are ubiquitous parts of modern life. It can be found in our smartphones, desktops, and smart home systems. To ensure its correctness in recognizing speeches, ASR needs to be tested. Testing ASR requires test cases in the form of audio files and their transcribed texts. Building these test cases manually, however, is tedious and time-consuming.**

To deal with the aforementioned challenge, in this work, we propose CrossASR, an approach that capitalizes the existing Text-To-Speech (TTS) systems to automatically generate test cases for ASR systems. CrossASR is a differential testing solution that compares outputs of multiple ASR systems to uncover erroneous behaviors among ASRs. CrossASR efficiently generates test cases to uncover failures with as few generated tests as possible; it does so by employing a failure probability predictor to pick the texts with the highest likelihood of leading to failed test cases. As a black-box approach, CrossASR can generate test cases for any ASR, including when the ASR model is not available (e.g., when evaluating the reliability of various third-party ASR services).

We evaluated CrossASR using 4 TTSes and 4 ASRs on the Europarl corpus. The experimented ASRs are Deepspeech, Deepspeech2, wav2letter, and wit. Our experiments on a randomly sampled 20,000 English texts showed that within an hour, CrossASR can produce, on average from 3 experiments, 130.34, 123.33, 47.33, and 8.66 failed test cases using Google, ResponsiveVoice, Festival, and Espeak TTSes, respectively. Moreover, when we run CrossASR on the entire 20,000 texts, it can generate 13,572, 13,071, 5,911, and 1,064 failed test cases using Google, ResponsiveVoice, Festival, and Espeak TTSes, respectively. Based on a manual verification carried out on statistically representative sample size, we found that most samples are actual failed test cases (audio understandable to humans but cannot be transcribed properly by an ASR), demonstrating that CrossASR is highly reliable in determining failed test cases. We also make the source code for CrossASR and evaluation data available at https://github.com/soarsmu/CrossASR.

***Index Terms*—Automatic Speech Recognition, Text-to-Speech, Test Case Generation, Differential Testing, Failure Probability Predictor**

I. Introduction

Automatic speech recognition (ASR) can be found anywhere we go. In our mobile devices, there are ASRs that we can use to type texts or communicate with a mobile virtual assistant (e.g., Siri[1] and Google Assistant[2]). Similar functionality is also available in our desktop PC (e.g., to speak with Cortana[3] virtual assistant in Windows PC). In our smart home systems, Alexa[4] has a built-in ASR to understand our instructions, such as turning off the light, playing music, etc.

The ubiquity of ASR makes it important to ensure that it works correctly. To test ASR, one needs a benchmark of audio files, which can be constructed by recording speeches, and their corresponding transcribed texts [1]. This process requires significant human effort and time.

Software testing is a crucial stage in a software development life cycle, where software quality is assessed [2]. The effectiveness of software testing is directly affected by the generation and selection of test cases [3]. Good test cases can be generated by a careful manual process, but such a process can be laborious and expensive. Also, due to the complexity of software systems, it is often easy for a tester to miss important test cases that can uncover failures. To reduce the cost and time in the testing process, and help uncover failures early, much research has been conducted on automated test generation [4].

Due to the rapid growth of AI and the proliferation of AI-powered systems, recently, many works have been proposed to test such systems, e.g., [5]–[8]. Although AI testing is a very active area, unfortunately, only a few are designed for ASRs. A recent survey by Zhang et al. [9], highlighted that: "The testing tasks currently tackled in the literature, primarily center on image classification. There remain open exciting testing research opportunities in many other areas, such as speech recognition, natural language processing, and agent/game play.". Due to the ubiquity of ASR, there is certainly a need for more work on ASR testing.

One of the most recent ASR testing work is DeepCruiser [10]. It is a pioneering approach that performs coverage-guided fuzzing using the idea of metamorphic testing. It applies several metamorphic transformations on audios, such as changing audio speed and volume, with the goal of generating audios that are incorrectly transcribed by ASRs. It performs an RNN abstraction and extracts the RNN's probabilistic transition model. It defines a stateful testing criterion and used it to guide the generation of tests for Deepspeech[5] ASR. It has the following limitations though: (1) It requires access to the ASR model. This kind of access may not always be possible, e.g., when the ASR is only available through APIs. Examples of such an ASR are Amazon

[1]https://www.apple.com/siri/
[2]https://assistant.google.com/
[3]https://www.microsoft.com/en-us/cortana/
[4]https://developer.amazon.com/en-GB/alexa
[5]https://github.com/mozilla/DeepSpeech

Transcribe[6] and Google Cloud Speech-To-Text[7]; (2) It can only generate test cases for RNN-based ASRs. There are ASRs developed using other techniques, including other deep neural network architectures, Hidden Markov Model, etc.; and (3) It also requires existing audio files and their corresponding transcribed texts as input.

In this work, we present CrossASR, which complements DeepCruiser and addresses its limitations. CrossASR is designed to *efficiently* generate test cases for ASRs by employing *differential testing* with *no requirement for manually labelled data* (i.e., transcribed texts of audio files). CrossASR achieves its efficiency by using a *failure probability predictor* to select texts that are likely to lead to failed test cases, saving time needed to generate and run many tests that do not lead to failures. CrossASR performs a *black-box* differential testing to uncover erroneous behaviors on ASR systems by cross-referencing different ASRs to detect different transcriptions from ASRs. CrossASR employs Text-to-Speech (TTS) engines to synthesize test inputs from texts and treats the texts as the ground truth labels, removing the need of manually transcribing audio files for getting labelled data.

Given a text collection, CrossASR works in multiple iterations. In the first iteration, CrossASR randomly selects a batch of texts from the collection. In the subsequent iterations, from the remaining texts in the collection, CrossASR utilizes a classifier trained from test cases generated in the previous iterations as a failure probability predictor. The classifier is used to pick the text having the highest probability of leading to failed test cases, thereby improving the efficiency of failed test case generation. CrossASR generates test cases until either a target number of failed test cases has been reached, a certain number of iterations has been completed, a time budget has been exhausted, or the whole text collection has been processed.

Each iteration of CrossASR proceeds as follows: CrossASR processes a batch of text; and each piece of text in the batch is fed to a TTS. The TTS synthesizes a speech audio, which is then fed to ASRs. ASRs output transcribed texts that they recognize. These ASR-transcribed texts are compared with the text that was input to the TTS. For each ASR, if the ASR-transcribed text matches with the text input to TTS, CrossASR determines the speech audio as a successful test case. If the ASR-transcribed text does not match the text input to TTS and there is another ASR's transcribed text that matches with the text input to TTS, CrossASR determines the speech audio as a failed test case. We consider cases where *all* ASRs' transcribed texts differ from the input text as indeterminable cases – as possibly the error resides in the TTS.

We evaluated the performance of CrossASR on 20,000 English texts (i.e., sentences) in the Europarl corpus [11], a corpus of parallel texts in 11 languages from the proceedings of the European Parliament[8]. We use 4 TTSes (i.e., Google[9], ResponsiveVoice[10], Festival[11], and Espeak[12]) and 4 ASRs (i.e., Deepspeech[13], Deepspeech2[14], wav2letter++[15], and wit[16]). We use more than one TTS to avoid bias that comes from a particular TTS. The experiment was ran separately for each TTS.

Our experiments show that, in terms of failed test case generation rate, CrossASR can generate, on average over 3 experiments, 130.34, 123.33, 47.33, and 8.66 failed test cases in the first hour using Google, ResponsiveVoice, Festival, and Espeak TTSes, respectively. When running on the entire 20,000 texts, CrossASR generates 13,572, 13,071, 5,911, and 1,064 failed test cases using Google, ResponsiveVoice, Festival, and Espeak TTSes, respectively. For the set of failed test cases, we take a random and statistically representative sample size. Our manual checks uncover that 96.96% of them are actual failed test cases, showing that CrossASR is highly reliable in determining failed test cases. Using the best TTS, the number of failed test cases generated for DeepSpeech, DeepSpeech2, wav2letter++, and wit are 4,036, 2,539, 2,202, and 4,795. Thus, CrossASR can also gauge the relative reliabilities of the ASRs.

The novel contributions of our work are as follows:

- We propose CrossASR, the first differential testing approach for ASRs. Prior approaches, including DeepCruiser, are based on metamorphic testing.
- Different from prior works, CrossASR does not require audio files and manually transcribed texts by leveraging TTSes.
- Also, CrossASR efficiently generate test cases for ASRs guided by a failure probability predictor. It differentiates itself from the many existing AI testing work that is coverage-guided, e.g., [5], [8], [12].
- We have evaluated CrossASR and showed that it can generate many failed test cases for 4 ASRs.

The rest of this paper is structured as follows. In Section II, we present our proposed approach. In Section III, we describe our dataset, experimental settings, and results. Section IV discusses sample generated test cases and false positives in the results. Section V presents related work. Finally, Section VI concludes the paper and mentions future work.

II. CrossASR

In this section, we present the overall architecture of CrossASR (Section II-A) and elaborate the details of its key component (Section II-B).

A. Architecture

CrossASR's architecture is shown at Figure 1. CrossASR accepts a *Text Collection* and returns a set of *Generated Test*

[6]https://aws.amazon.com/transcribe/
[7]https://cloud.google.com/speech-to-text
[8]http://www.statmt.org/europarl/
[9]https://cloud.google.com/text-to-speech
[10]https://responsivevoice.org/
[11]http://www.cstr.ed.ac.uk/projects/festival/
[12]http://espeak.sourceforge.net/
[13]https://github.com/mozilla/DeepSpeech
[14]https://github.com/PaddlePaddle/DeepSpeech
[15]https://github.com/facebookresearch/wav2letter
[16]https://wit.ai/

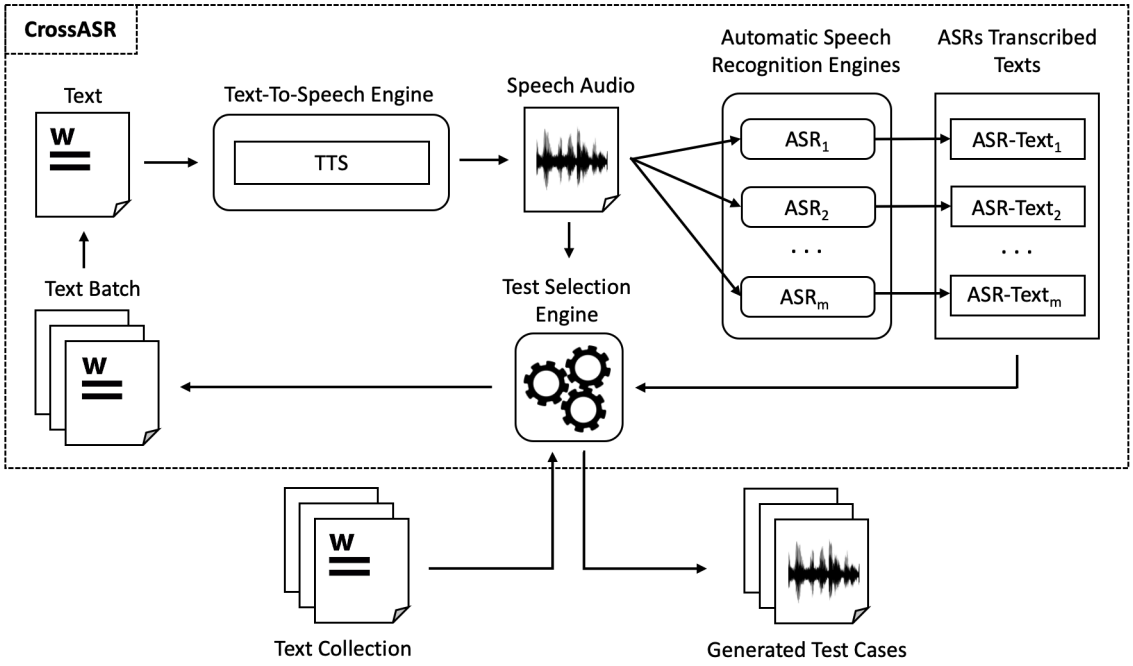

Fig. 1. Architecture of CrossASR

Cases. Given a *Text Collection*, the *Test Selection Engine* selects a *Text Batch*. Each *Text* in the *Text Batch* is then fed into the *Text-To-Speech Engine (TTS)*. The TTS converts the *Text* into a *Speech Audio*. The *Speech Audio* is then fed to the *Automatic Speech Recognition (ASR) Engines*. These engines consist of *m* ASRs, thereby producing m texts for each speech audio. In total, for each batch with *n* number of texts, there are $n \times m$ transcribed texts, which we refer to as *ASRs Transcribed Texts*.

Test Selection Engine performs differential testing by cross-referencing *ASRs Transcribed Texts* to detect incorrect behaviors of ASRs. It compares the *ASRs Transcribed Texts* with the input *Text* and considers three possible cases:

Case 1: *All ASR transcribed texts matches the input Text.*

In this case, there is a low likelihood that the generated speech audios do not match with the input *Text*. Therefore, CrossASR considers all ASRs to have successfully transcribed the audio. CrossASR then generates m 5-tuples, each corresponding to a successful test case for one of the m ASRs. Each 5-tuple is of the format $\langle t, s, a, t', l \rangle$, where t is the input text, s is the TTS generated audio, a is the ASR used to transcribe t, t' is the ASR transcribed text, and l is the status of the test case (in this case: success). The m 5-tuples are put in the set of *Generated Test Cases*.

Case 2: *At least one of the ASR transcribed texts that matches with the input Text while at least one of the other ASR Transcribed Texts do not match with the input Text.*

In this case, CrossASR considers cases where the transcribed text does not match input text as failed test cases; it considers the other cases as successful test cases. It will generate the corresponding m 5-tuples $\langle t, s, a, t', l \rangle$, whereas l = success (iff $t \neq t'$) or failed (otherwise). These test cases are put in the set of *Generated Test Cases*.

Case 3: *All ASR transcribed texts do not match with the input Text.*

This may occur due to the limitation of the TTS. Therefore, CrossASR cannot confidently determine whether the test cases are successful or failed test cases. CrossASR considers all m 5-tuples corresponding to such test cases as *Indeterminable Test Cases* and also put them in the collection of Generated Test Cases.

B. Test Selection Engine

Test Selection Engine has two different workflows depending on the availability of *Generated Test Cases*. The first workflow is used when there are no available test cases, which we refer to as *Test Selection Engine's First Iteration*. The second workflow is used when there are available test cases, which we refer to as *Test Selection Engine's Subsequent Iterations*. We present the detail of these workflows below:

Test Selection Engine's First Iteration

The *Test Selection Engine's First Iteration* workflow is shown in Figure 2. The numbers in the figure specify the execution order.

Steps 1-3: The *Batch Feeder* takes a batch of *n* randomly sampled texts from the *Text Collection* and feed it to the *Text Preprocessor*. The *Text Preprocessor* lowercases the text, removes punctuation from it, and normalizes non-standard words (e.g., abbreviations, numbers, date, and currency expressions). The text normalization algorithm is based on Flint et al.'s

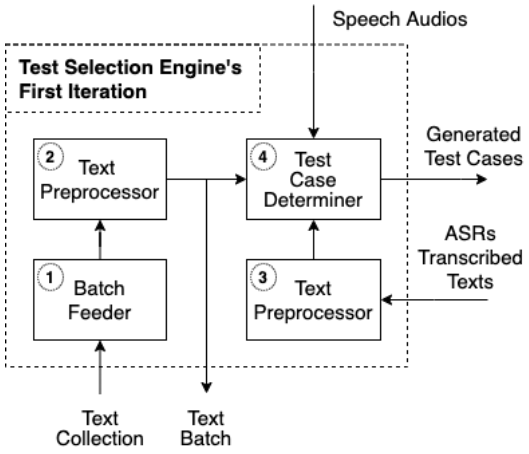

Fig. 2. Workflow of the *Test Selection Engine* for the First Iteration

work [13]. This normalization is intended to ensure that the text are pronounced properly by TTS. One of the normalization step is a rule-based substitution. We find the rules insufficient, therefore we added a few more rules. For example, we add a rule that replaces the word "mr" to "mister" and "mrs" to "missus".

Step 4: The preprocessed batch of n texts are output as *Text Batch*. *Text Batch*, its corresponding *Speech Audios*, and *ASRs Transcribed Texts* that has been preprocessed by *Text Preprocessor* are then fed into *Test Case Determiner*. *Test Case Determiner* categorizes each test case as either a *Success Test Case*, a *Failed Test Case*, or an *Indeterminable Test Case*. The result is a set of 5-tuples that is output as *Generated Test Cases*.

Test Selection Engine's Subsequent Iterations

The workflow of *Test Selection Engine*'s subsequent iterations is shown in Figure 3. The numbers in the figure specify the execution order.

Steps 1-5: The *Classifier Builder* trains a classifier using the *Generated Test Cases* from the previous iterations. The *Batch Feeder* takes a batch of n randomly selected texts from the *Text Collection* (that were not included in earlier batches), and feed it to the *Text Preprocessor*. Texts are preprocessed similarly like in *Test Selection Engine's First Iteration*. The *Text Selector* utilizes the trained classifier as a *failure probability predictor* to select texts from the batch according to their likelihood of leading to failed test cases. The selected texts are output as *Text Batch*, which would be fed to the TTS.

Step 6: Next, *Text Preprocessor* preprocessed *ASR Transcribed Texts* and input it to the *Test Case Determiner*, which categorizes each test case into a *Success Test Case*, a *Failed Test Case*, or an *Indeterminable Test Case*. The result is a set of test cases (in the form of 5-tuples) that is added to the *Generated Test Cases*. At this point, the updated *Generated Test Cases* are used to re-train the classifier in the next iteration and the whole process is repeated until a target number of *Failed Test Cases* has been reached, a certain number of iterations has been completed, a time budget has been exhausted, or the whole *Text Collection* has been explored.

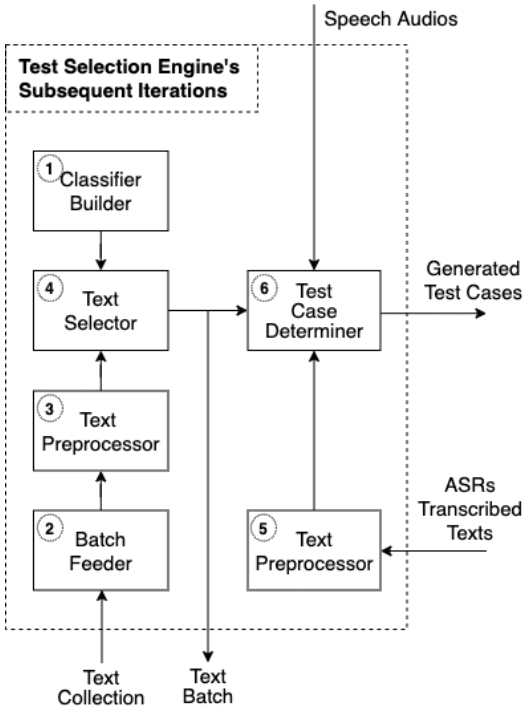

Fig. 3. Workflow of the Test Selection Engine for Subsequent Iterations

The *Classifier Builder* and *Text Selector* deserve further elaboration, which is provided below. *Classifier Builder* constructs a two-stage classifier given *Generated Test Cases*, which is a set of 5-tuples. In the first stage, the classifier is trained using the *Generated Test Cases* to classify a text into two labels: determinable (success or failed) vs. indeterminable. In the second stage, the classifier is trained using failed and successful test cases in *Generated Test Cases* to classify a text into one that likely leads to failed test cases and one that likely leads to successful test cases. For this training, a text has m labels, each corresponds to an ASR. The text is only labeled as one that likely leads to successful test cases if and only if all m labels are success. The *Text Selector* uses the first stage classifier to select texts that are likely to lead to determinable (failed or success) test cases.

The *Text Selector* picks only the top-k text from the ranked list of selected texts sorted in descending order of their likelihood to lead to determinable test cases. From these top-k texts, the *Text Selector* then uses the second stage classifier to select texts that are likely to lead to failed test cases. We use the probability outputs by the second stage classifier as the *failure probability*.

CrossASR utilizes ALBERT [14], a Transformer-based deep neural network, as its classifier since Transformer [15] has shown state-of-the-art performances for NLP tasks in recent years. ALBERT is an upgrade of BERT [16] that advances the state-of-the-art performance on several NLP tasks including

text classification. We use the ALBERT pre-trained model to reduce the training time. This pre-trained model has been trained on a large set of unlabelled texts and the knowledge from this set can be transferred to our task when training using our labelled texts, thereby speeding up the training time.

III. Experiments

In this section, we describe the dataset and the experimental settings used in our experiments. We also present our research questions and findings. We end the section by discussing threats to validity.

A. Dataset

Our dataset is taken from Europarl [11], a corpus of parallel texts in 11 languages from the proceedings of the European Parliament[17]. We collect all English texts in the corpus and discard texts from other languages. These texts have been carefully segmented and aligned using Church and Gale Algorithm. In total, there are 22,683,244 English texts. After dropping the empty and duplicate texts, we are left with 2,398,750 texts. We randomly pick 20,000 texts for our experiment.

B. Experimental Settings

Our experiments are performed on a desktop computer running Ubuntu 18.04 OS with Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz processor, 64GB RAM, and NVIDIA GeForce RTX 2080. We experiment with TTSes and ASRs listed in Tables I and II, respectively. We set n to 210, which is an average number of texts that can be processed within an hour, and k to 400. We use SimpleTransformers [18] library to implement the ALBERT classifier. Specifically, we use albert-base-v2 pre-trained model[19].

TABLE I
TTSes in Our Experiment

Name	URL
GoogleTTS	https://cloud.google.com/text-to-speech
Festival	http://www.cstr.ed.ac.uk/projects/festival/
ResponsiveVoice	https://responsivevoice.org/
Espeak	http://espeak.sourceforge.net/

TABLE II
ASRs in Our Experiment

Name	URL
Deepspeech [17]	https://github.com/mozilla/DeepSpeech
DeepSpeech2 [18]	https://github.com/PaddlePaddle/DeepSpeech
wav2letter [19]	https://github.com/facebookresearch/wav2letter
wit	https://wit.ai/

[17]http://www.statmt.org/europarl/
[18]https://simpletransformers.ai/
[19]https://huggingface.co/transformers/pretrained_models.html

C. Research Questions

RQ1: *How many failed test cases can be generated by CrossASR in the first iteration?*

In this research question, we measure the failed test case generation rate. To do so, we run CrossASR for one iteration (one hour) and count the total number of failed test cases generated. Due to the randomness of initial texts selection, the experiments are repeated 3 times and the reported metrics are averaged.

RQ2: *How useful is the Failure Probability Predictor (FPP) in CrossASR?*

We investigate how efficient CrossASR in generating failed test cases if it uses the FPP. To do so, we run CrossASR for 5 iterations and count the number of failed test case obtained for each iteration. We compare the number of failed test cases generated by CrossASR with and without FPP. Due to the randomness of initial texts selection, the experiments are repeated 3 times and the reported metrics are averaged.

RQ3: *How many failed test cases can be generated by CrossASR after processing the entire 20,000 texts?*

We run CrossASR on the entire 20,000 texts for all combinations of ASRs and TTSes. We then count the total number of failed test cases that is generated from these 20,000 texts.

RQ4: *How accurate is CrossASR in determining whether a test case is a failed test case?*

CrossASR analyzes the differences between the text input to TTS and its corresponding ASR-transcribed texts to determine whether a test case is a *Success Test Case*, *Failed Test Case*, and *Indeterminable Test Case*. To measure the reliability of CrossASR in determining failed test cases, we ask a native English speaker, who is not an author of this paper to verify whether the failed test cases generated by CrossASR are actual failed test cases. We then calculate the *False Positive Rate (FPR)*, which is the proportion of the failed test cases that are incorrect (e.g., they are actually not failed test cases).

TABLE III
Number of Failed Test Cases Generated in the First Iteration (an Hour)

TTS	DS	DS2	W2L	Wit	Total
Google	38.00	25.67	21.67	45.00	**130.34**
RV	44.33	23.33	15.67	39.00	**123.33**
Festival	15.33	9.67	8.33	14.00	**47.33**
Espeak	2.67	2.33	0.33	3.33	**8.66**
Total	100.33	61.00	46.00	101.33	**308.66**

RV: ResponsiveVoice
DS: Deepspeech
W2L: Wav2Letter++

D. Results

RQ1: Average Number of Failed Test Cases Generated in The First Iteration

Table III shows the number of failed test cases generated for each combination of TTSes and ASRs in the first hour of running CrossASR. Using Google, ResponsiveVoice, Festival, and Espeak TTSes, our CrossASR generates an average of 130.34, 123.33, 47.33, and 8.66 failed test cases in the first iteration (1 hour).

If we look from the point of view of ASRs, Deepspeech, Deepspeech2, Wav2letter++, and Wit has 100.33, 61.00, 46.00, and 101.33 generated failed test cases in the first hour. The number of failed test cases can be used to measure the reliability of ASR in recognizing speech audios. If the number of the failed test case generated for an ASR is low, the ability of the ASR in recognizing speech audios is high. If the number of the failed test case generate for an ASR is high, the ability of the ASR in recognizing speech audios is low. From this perspective, Wit can be considered as the worst ASR among the 4 since it has the highest number of failed test cases. Wav2Letter++ can be considered as the best ASR since it has the lowest number of failed test cases.

RQ2: Average Number of Failed Test Cases Generated for 5 Iterations Running CrossASR with and without FPP

Figure 4 shows a detailed comparison of the number of failed test cases generated for every iteration with and without FPP. The graph shows that CrossASR without FPP generates less failed test cases for all combinations of TTSes and ASRs, thereby demonstrating the usefulness of FPP. The largest drops in number of generated failed test cases can be observed for DeepSpeech and wit, while the smallest drops can be observed for wav2letter.

Figure 5 shows the cumulative number of failed test cases using 4 TTSes across 4 ASRs. FPP is successful in selecting texts that lead to failed test cases, as demonstrated by the higher number of failed test cases generated over time when FPP is used. Table IV shows the speed up with FPP in finding failed test cases at the end of the 5th iteration. Our experiment shows that for each TTS, by using FPP, the minimum speed up of the failed test case generation rate is 2.35 times. FPP achieves the best speed up for Espeak, with 14.91 times speed up, generating 655.6 failed test cases in 5 hours.

TABLE IV
THE SPEED-UP OF CROSSASR IN FINDING FAILED TEST CASES WITH FPP

TTS	# Number of Failed Test Case		Speed Up
	With FPP	Without FPP	
Google	1,566.67	665.68	**2.35**
ResponsiveVoice	1,886.01	635.32	**2.97**
Festival	1,403.31	198.34	**7.08**
Espeak	655.6	43.98	**14.91**

RQ3: Number of Failed Test Cases Generated from the Entire 20,000 Texts

Table V shows the total number of failed test cases that is generated from the entire 20,000 texts for all combinations of experimented TTSes and ASRs. Using 4 TTses and 4 ASRs, there will be a total 320,000 test cases generated in the form of failed test cases, success test cases and indeterminable test cases. Our experiment showed that CrossASR generates 13,572, 13,071, 5,911, and 1,064 failed test cases for all ASRs when using Google, ResponsiveVoice, Festival, and Espeak TTSes, respectively. Google TTS produces the highest number of failed test cases overall and for each of the experimented ASRs. It produces 4,036, 2,539, 2,202, and 4,795 failed test cases for DeepSpeech, DeepSpeech2, wav2letter++, and wit, respectively. On average, for each 100 processed texts, it can produce 16.96 failed test cases for each ASRs. Espeak is the worst among others. On average, for each 100 processed texts, it can produce only 1.33 failed test cases for each ASRs.

TABLE V
NUMBER OF FAILED TEST CASES GENERATED FOR EACH COMBINATION OF TTSES AND ASRS

TTS	# Failed Test Case				Total
	DS	DS2	W2L	Wit	
Google	4,036	2,539	2,202	4,795	**13,572**
RV	4,510	2,436	1,911	4,214	**13,071**
Festival	1,819	1,229	1,035	1,908	**5,911**
Espeak	359	326	44	335	**1,064**
Total	**10,724**	**6,530**	**5,192**	**11,252**	**33,618**

RV: ResponsiveVoice
DS: Deepspeech
W2L: Wav2Letter++

RQ4: Accuracy of CrossASR in Determining Failed Test Cases for Each ASR

The effectiveness of CrossASR in determining failed test cases is shown in the Table VI. For each ASR, we asked a native speaker non-author to verify whether the ASR transcribed text matches with the speech input to ASR while the speech input is matched with the ground truth text generated from *Text Collection*. We take a random sample of failed test cases with a 95% confidence level and 5% confidence interval. The non-author verified that in the random sample, CrossASR generates failed test cases with the highest and lowest FPR of 9% and 0%, respectively. From all combination of TTSes and ASRs, the FPRs are low, which means that the failed test cases that are detected by CrossASR actually highlight a deficiency in the corresponding ASR (it is a true positive). It shows that CrossASR is reliable in determining failed test cases for ASRs.

E. Threats to Validity

Internal Validity. The results for RQ2 and RQ3 depend upon the randomness of selecting a batch of texts from the *Text Collection*. It is possible that the results are different from randomly selected texts. To mitigate this threat, we rerun the experiment 3 times with different randomizations. We report the average results across the three experiments.

External Validity. Text-to-Speech is a crucial part of CrossASR system. The quality of a generated speech by TTS affects the number of test cases generated. Instead of running the experiment using one TTS, we run on 4 TTSes to minimize a potential bias that come from the usage of one TTS. The result shows that for 4 TTSes used in our experiment, CrossASR is able to generate highly reliable failed test cases automatically.

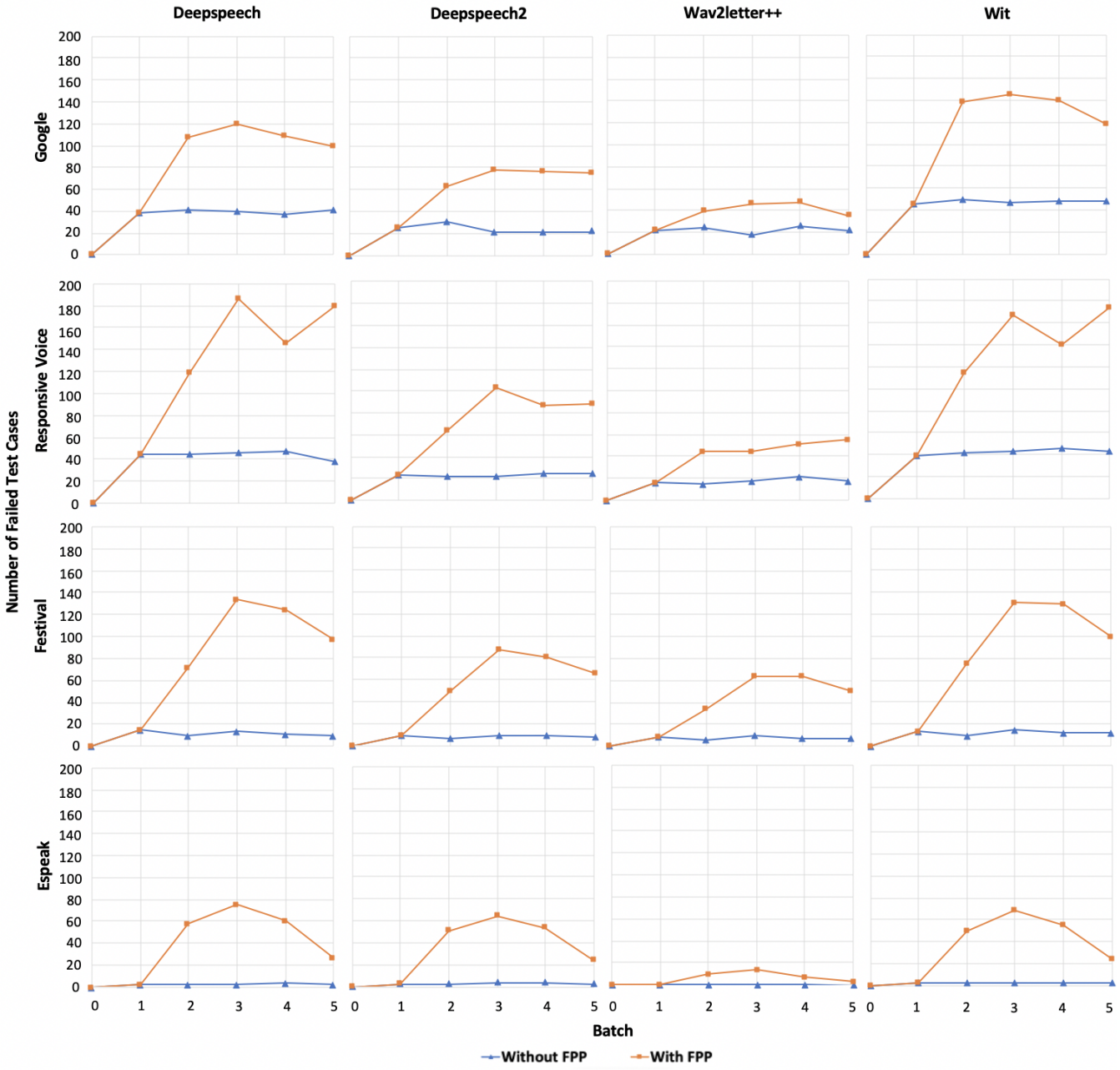

Fig. 4. Comparison of The Number of Failed Test Cases Generated by CrossASR (with and without FPP). A row indicates the number of failed test cases generated using a TTS for different ASRs (the columns).

TABLE VI
ACCURACY OF CROSSASR IN DETERMINING FAILED TEST CASES FOR EACH ASR

TTS	Deepspeech			Deepspeech2			wav2letter++			Wit		
	# Sample	# FP	FPR	# Sample	# FP	FPR	# Sample	# FP	FPR	# Sample	# FP	FPR
Google	47	3	6%	30	0	0%	26	1	3%	55	0	0%
ResponsiveVoice	52	1	1%	28	1	3%	22	0	0%	51	2	9%
Festival	21	1	4%	15	0	0%	12	1	8%	22	1	4%
Espeak	5	0	0%	4	0	0%	1	0	0%	4	0	0%

FP: False Positive
FPR: False Positive Rate

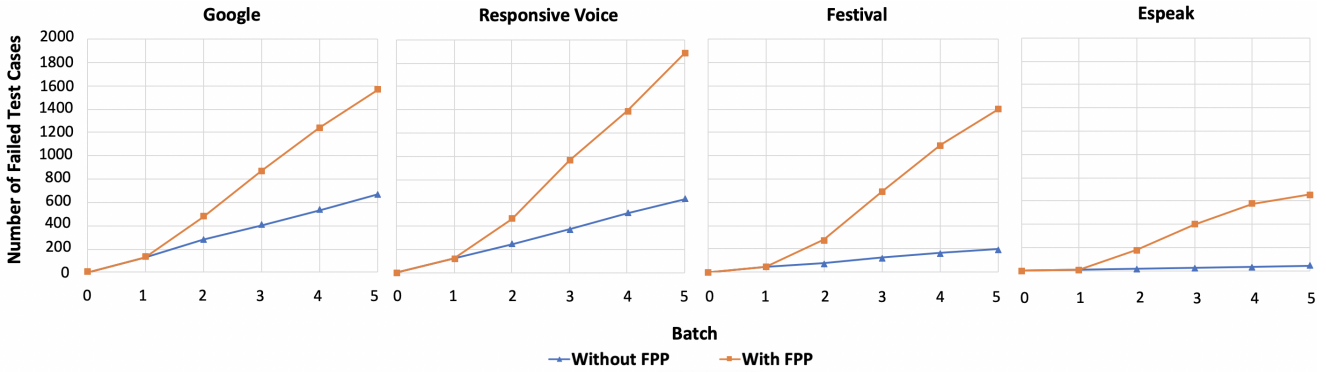

Fig. 5. Comparison of The Cumulative Number of Failed Test Cases Generated Across ASRs (with and without FPP) when using different TTSes

IV. Discussion

In this section, we provide examples of failed test cases generated by CrossASR. We also analyze false positive failed test cases found by our manual verification.

A. Examples of Success Test Case, Failed Test Case, and Indeterminable Test Case

Our dataset contains 20,000 texts. Each text is used to generate 4 test cases, each corresponds to an ASR. Since we use 4 TTSes, we will have a total of 320,000 generated test cases. Table VII shows the number of test cases generated using each TTS. Among these test cases, 33,618, 25,806, and 260,576 of them are failed, successful, and indeterminable test cases, respectively.

TABLE VII
Number of Test Cases Generated

TTS	# FTC	# STC	# ITC
Google	13,572	11,128	55,300
ResponsiveVoice	13,071	10,049	56,880
Festival	5,911	4,213	69,876
Espeak	1,064	416	78,520
Total	**33,618**	**25,806**	**260,576**

FTC: Failed Test Case
STC: Success Test Case
ITC: Indeterminable Test Case

Table VIII shows an example of a success test case for all ASRs. Tables IX, X, XI, and XII show examples of failed test cases for DeepSpeech, DeepSpeech2, wav2letter++, and wit respectively. DeepSpeech mistakenly transcribed "consulted" to "controlled". DeepSpeech2 mistakenly transcribed "and the mess" to "in the met". Wav2letter++ mistakenly transcribed "transport" to "trains". Wit mistakenly transcribed "an old objective" to "another project".

TABLE VIII
An Example of Texts That are a Success Test Case

Type	Text
Original Text	they have both been guardians of the treaty as well as guardians of the general interests of europe
Human Transcribed Text	they have both been guardians of the treaty as well as guardians of the general interests of europe

TABLE IX
An Example of Texts from a Failed Test Case for DeepSpeech

Type	Text
Original Text	they can be consulted without difficulty
ASR Transcribed Text	they can be controlled without difficulty
Human Transcribed Text	they can be consulted without difficulty

TABLE X
An Example of Texts for a Failed Test Case for DeepSpeech2

Type	Text
Original Text	as far as this is concerned it is up to both the commission and the mess to make this even clearer to these consumers
ASR Transcribed Text	as far as this is concerned it is up to both a commission in the met to make this even clearer to these consumers
Human Transcribed Text	as far as this is concerned it is up to both the commission and the mess to make this even clearer to these consumers

TABLE XI
An Example of Texts for a Failed Test Case for Wav2letter++

Type	Text
Original Text	the proposed maximum transport time however is not totally adequate
ASR Transcribed Text	the proposed maximum trains time however is not totally adequate
Human Transcribed Text	the proposed maximum transport time however is not totally adequate

TABLE XII
An Example of Texts for a Failed Test Case for Wit

Type	Text
Original Text	a new strategy for an old objective
ASR Transcribed Text	a new strategy for another project
Human Transcribed Text	a new strategy for an old objective

Table XIII shows an example of an indeterminable test case. DeepSpeech mistakenly transcribes "estimates" to "at the mates" and "could" to "would". DeepSpeech2 mistakenly transcribe "income among" to "intermont". Wav2letter++ mistakenly transcribe "that" to "them". Wit mistakenly transcribes "losses of income among" to "loss of income mile".

TABLE XIII
AN EXAMPLE OF TEXTS FROM AN INDETERMINABLE TEST CASE

Type	Text
Original Text	experts estimate that making such an agreement could result in very large losses of income among farmers
Deepspeech Transcribed Text	experts at the mates that making such an agreement would result in very large losses of income among farmers
Deepspeech2 Transcribed Text	experts estimate that making such an agreement could result in very large losses of intermont farmers
Wav2letter++ Transcribed Text	experts estimate them making such an agreement could result in very large losses of income among farmers
Wit Transcribed Text	experts estimate that making such an agreement could result in very large loss of income mile farmers
Human Recognized Text	experts estimate that making such an agreement could result in very large losses of income among farmers

B. Analysis of False Positive Failed Test Case from Manual Verification

From 33,618 failed test cases, we manually verified a statistically representative sample of size 395 and found 12 of them to be false positives. We analyze these false positives and observed that they can be categorized into 4 different cases:

(1) Human deems the audio to be the same as the transcribed text, but different from the original text. This highlights a problem with the TTS.

Original Text
so we need more europe and a european policy which **produces** tangible benefits throughout the european continent

ASR Transcribed Text
so we need more europe and a european policy which **reduces** tangible benefits throughout the european continent

The speech audio is matched with the word "reduces" in the transcribed text, instead of the word "produces" in the original text. This may be triggered by the TTS when generating the speech. This case is found in an experiment using Google TTS and Deepspeech ASR.

(2) Human deems the audio to be different from both transcribed and input text. This indicates a problem with the TTS.

Original Text
chorus is due to be accepted as a new member state of the european union on one may two thousand and four

ASR Transcribed Text
corridor to be accepted as a new member state of the european union on one mat thousand and four

This problem may be caused by the TTS inability in generating uncommon words such as "chorus" in the original text. We found this case in ResponsiveVoice and Festival.

(3) Cases highlighting limitations of our Text Preprocessing step. The text normalization substep of our text processing step is not perfect. Sometimes there is just a small syntactic difference between the original and ASR transcribed text.

Original Text
this is obviously important because we are spending **taxpayers** money

ASR Transcribed Text
this is obviously important because we are spending **tax payers** money

This example occured in Wav2letter++ where the word "taxpayers" is transcribed to "tax payers". Another example occured in Wit, the word "may be" is transcribed to "maybe". To handle this problem, we make use of a previous work by Flint et al. on text normalization. It uses a rule-based substitution to standardize the words as described in Section II. We add a few more rules to such as changing the word "mr" to "mister" and "mrs" to "missus". However, there are no rules for the words mentioned above. The correctness of CrossASR in detecting a failed test case relies on checking for perfect syntactic matches. This causes false positives for texts that are semantically equivalent but have minor syntactic variations (e.g., words that can appear in several forms). We can improve our text-processing step to address this issue.

(4) Audio is ambiguous; It can match both texts. For this case, by listening to the audio, the original text and ASR transcribed text can not be differentiated from each other easily. The speech articulation is similar to both original text and ASR transcribed text. We found two instances of this case. Both involve Google TTS and Wav2letter++ ASR. In this example, the word "socioeconomic" in the original text sounds similar to the word "social economic" in the ASR transcribed text.

Original Text
i entirely agree with all those who have talked today about the need to pursue a policy of peace by economic means as well and to remove the **socioeconomic** causes of instability and i think that this is a field where the european union is already doing a great deal for the outside world

ASR Transcribed Text
i entirely agree with all those who have talked to day about the need to pursue a policy of peace by economic means as well and to remove the **social economic** causes of instability and i think that this is a field where the european union is already doing a great deal for the outside world

V. RELATED WORK

In this section, we present the previous works on ASR testing and AI testing.

A. ASR Testing

Previous works on generating test cases for ASR came from both the software engineering community and the artificial intelligence community. To the best of our knowledge, DeepCruiser [10] was the only work from the software engineering community. We have presented DeepCruiser and its differences with CrossASR in Section I.

From the artificial intelligence community, there are several works that perform adversarial attacks targeting ASR systems (a.k.a. adversarial test case generation) [20]–[25]. Similar to DeepCruiser, these approaches utilize metamorphic relation as test oracle; specifically, they produce audio files by applying imperceptible perturbations to existing audio files. Also, like DeepCruiser, these approaches require existing audio files and their corresponding transcribed texts as input. Moreover, these approaches test *robustness* of ASRs (its ability to produce the same output in the face of minor perturbations). Different from these works, CrossASR performs differential testing by cross-referencing different ASRs as test oracle. Also, CrossASR is concerned in assessing *correctness* of ASRs instead of their robustness. Robustness and correctness (aka. accuracy) have been shown to be two competing goals; and both are important, c.f., [26].

B. AI Testing

There are many works on AI testing. Here, due to page limit, we only highlight a few. For a a comprehensive treatment of the subject, please refer to the survey by Zhang et al. [9].

Test Input Generation. DeepXplore [5] originally introduces neuron coverage for systematically measuring the parts of a deep learning system exercised by test inputs. DeepConcolic [27] proposes a white-box deep learning testing technique to generate inputs based on concolic testing. DeepStellar [8] proposes a set of metrics for recurrent neural networks based on state modeling, which are applied to adversarial sample detection and coverage-guided test input generation. DeepHunter [12] proposes a metamorphic mutation strategy to generate new semantically preserved tests, and leverage multiple extensible coverage criteria as feedback to guide the test generation. DeepCheck [28] generates inputs for text classification by using a fuzzing approach that considers the text grammar and the distance between inputs. Sun et al. [29] tested an automatic translation system by mutating words in translation inputs. Different than the above approaches, CrossASR generates inputs specifically for testing ASRs – in the form of speech audios and their corresponding texts.

Test Oracles. Several popular types of test oracles have been studied for AI testing, e.g., metamorphic relations and cross-referencing [9]. Metamorphic relations transform training or test data to yield certain expected changes in the output prediction [30]. Metamorphic relation was applied in real-world problems, e.g., autonomous driving [7], [31]. For example, DeepTest [7] automatically generates test cases leveraging real-world changes in driving conditions like rain, fog, and lighting conditions. DeepRoad [31] automatically synthesized large amounts of diverse driving scenes with various weather conditions.

Cross-referencing as test oracle has been used in differential testing and N-version programming. For example, CRADLE [32] finds and localizes bugs in the implementations of deep learning models by cross-checking multiple backends, i.e. TensorFlow, CNTK, and Theano. DiffChaser [33] is an automated black-box differential testing technique to detect disagreements between version variants of a DNN. DLFuzz [6] uses predicted difference between original and mutated inputs as test oracles. Qin et al. [34] synthesized a mirror program and used its behaviour as pseudo oracles. Differential testing in ASR has not been studied yet. CrossASR is the first to use differential testing to detect failed test cases in ASR systems. Differing transcriptions among multiple ASR systems are an indicator of bug.

VI. Conclusion and Future Work

In this work, we propose CrossASR, an approach to automatically generate test cases for ASR systems using text-to-speech engines. CrossASR performs differential testing by cross-referencing multiple ASR systems. In addition, CrossASR employs a failure probability predictor to efficiently guide the test case generation for ASR systems. In terms of efficient generation of failed test cases, within an hour, CrossASR can produce, on average, 130.34, 123.33, 47.33, and 8.66 failed test cases using Google, ResponsiveVoice, Festival, and Espeak TTSes, respectively. Our experiment on running CrossASR on 20,000 texts showed that CrossASR can generate 13,572, 13,071, 5,911, and 1,064 failed test cases using Google, ResponsiveVoice, Festival, and Espeak TTSes, respectively. CrossASR is highly reliable in determining failed test cases, as demonstrated by manual verification. Almost all of the failed test case samples inspected are actual failed test cases (audio understandable to humans but cannot be transcribed properly by an ASR).

In the future, we plan to evaluate CrossASR using additional ASRs and TTSes. We also plan to experiment using another text dataset, including software engineering (SE) dataset such as StackOverflow, GitHub comments, GitHub source code, and software bug reports. SE datasets can be used to determine the reliability of current ASRs for speech-based programming environments [35], [36]. We want to add some metamorphic transformations to generate more speech audios that can lead to failed test cases. Additionally, we plan to design a technique that can synthesize new text (rather than selecting from a text collection) that likely leads to failed test cases. Moreover, it would be interesting to investigate ways to repair ASRs given a set of failed test cases.

Replication Package. The source code for CrossASR is available at https://github.com/soarsmu/CrossASR

Acknowledgment. This work was supported by Lee Kuan Yew Fellowship awarded by Singapore Management University.

REFERENCES

[1] D. S. Pallet, W. M. Fisher, and J. G. Fiscus, "Tools for the analysis of benchmark speech recognition tests," in *International Conference on Acoustics, Speech, and Signal Processing*, 1990, pp. 97–100 vol.1.

[2] P. Roger S, *Software Engineering A Practitioners Approch*. Mc Graw Hill International Edition, 2010.

[3] M. Dave and R. Agrawal, "Mutation testing and test data generation approaches: A review," in *Smart Trends in Information Technology and Computer Communications*, A. Unal, M. Nayak, D. K. Mishra, D. Singh, and A. Joshi, Eds. Singapore: Springer Singapore, 2016, pp. 373–382.

[4] N. Jain and R. Porwal, "Automated test data generation applying heuristic approaches—a survey," in *Software Engineering*, M. N. Hoda, N. Chauhan, S. M. K. Quadri, and P. R. Srivastava, Eds. Singapore: Springer Singapore, 2019, pp. 699–708.

[5] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *26th Symposium on Operating Systems Principles (SOSP)*, 2017, p. 1–18.

[6] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2018, pp. 739–743.

[7] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," *IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 303–314, 2018.

[8] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, "DeepStellar: Model-based quantitative analysis of stateful deep learning systems," in *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2019, p. 477–487.

[9] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

[10] X. Du, X. Xie, Y. Li, L. Ma, J. Zhao, and Y. Liu, "DeepCruiser: Automated guided testing for stateful deep learning systems," *ArXiv*, vol. abs/1812.05339, 2018.

[11] K. Philipp, "Europarl: A parallel corpus for statistical machine translation," in *Machine Translation Summit X*, 2005, pp. 79–86.

[12] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: A coverage-guided fuzz testing framework for deep neural networks," in *28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2019, p. 146–157.

[13] E. J. Flint, E. Ford, O. Thomas, A. Caines, and P. Buttery, "A text normalisation system for non-standard english words," in *NUT@EMNLP*, 2017.

[14] L. Zhenzhong, C. Mingda, G. Sebastian, G. Kevin, S. Piyush, and S. Radu, "Albert: A lite bert for self-supervised learning of language representations," in *International Conference on Learning Representation (ICLR 2020)*, 2020.

[15] V. Ashish, S. Noam, P. Niki, U. Jakob, J. Llion, G. Aidan N., K. Lukasz, and P. Illia, "Attention is all you need," in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.

[16] D. Jacob, C. Ming-Wei, L. Kenton, and T. Kristina, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.

[17] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep Speech: Scaling up end-to-end speech recognition," *ArXiv*, vol. abs/1412.5567, 2014.

[18] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. Hannun, T. Han, L. Johannes, B. Jiang, C. Ju, B. Jun, P. LeGresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, Y. Xie, D. Yogatama, B. Yuan, J. Zhan, and Z. Zhu, "Deep Speech 2: End-to-end speech recognition in english and mandarin," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 173–182. [Online]. Available: http://proceedings.mlr.press/v48/amodei16.html

[19] V. Pratap, A. Hannun, Q. Xu, J. Cai, J. Kahn, G. Synnaeve, V. Liptchinsky, and R. Collobert, "Wav2Letter++: A fast open-source speech recognition system," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6460–6464.

[20] N. Carlini and D. Wagner, "Audio adversarial examples: Targeted attacks on speech-to-text," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018, pp. 1–7.

[21] L. Schönherr, K. Kohls, S. Zeiler, T. Holz, and D. Kolossa, "Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding," *ArXiv*, vol. abs/1808.05665, 2019.

[22] Y. Qin, N. Carlini, I. J. Goodfellow, G. W. Cottrell, and C. Raffel, "Imperceptible, robust, and targeted adversarial examples for automatic speech recognition," in *ICML*, 2019.

[23] S. Khare, R. Aralikatte, and S. Mani, "Adversarial black-box attacks for automatic speech recognition systems using multi-objective genetic optimization," *ArXiv*, vol. abs/1811.01312, 2018.

[24] H. Abdullah, M. S. Rahman, W. Garcia, L. Blue, K. M. Warren, A. Yadav, T. Shrimpton, and P. Traynor, "Hear "no evil", see "kenansville": Efficient and transferable black-box attacks on speech recognition and voice identification systems," *ArXiv*, vol. abs/1910.05262, 2019.

[25] R. Taori, A. Kamsetty, B. Chu, and N. Vemuri, "Targeted adversarial examples for black box audio systems," in *2019 IEEE Security and Privacy Workshops (SPW)*, 2019, pp. 15–20.

[26] D. Su, H. Zhang, H. Chen, J. Yi, P. Chen, and Y. Gao, "Is robustness the cost of accuracy? - A comprehensive study on the robustness of 18 deep image classification models," in *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XII*, 2018, pp. 644–661.

[27] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "DeepConcolic: Testing and debugging deep neural networks," in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*, ser. ICSE '19. IEEE Press, 2019, p. 111–114.

[28] S. S. Udeshi and S. Chattopadhyay, "Grammar based directed testing of machine learning systems," *IEEE TSE*, 2019.

[29] Z. Sun, J. M. Zhang, M. Harman, M. Papadakis, and L. Zhang, "Automatic testing and improvement of machine translation," in *IEEE/ACM 42th International Conference on Software Engineering (ICSE)*, 2020.

[30] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepmutation: Mutation testing of deep learning systems," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, 2018, pp. 100–111.

[31] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 132–142. [Online]. Available: https://doi.org/10.1145/3238147.3238187

[32] H. V. Pham, T. Lutellier, W. Qi, and L. Tan, "Cradle: Cross-backend validation to detect and localize bugs in deep learning libraries," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE '19. IEEE Press, 2019, p. 1027–1038.

[33] X. Xie, L. Ma, H. Wang, Y. Li, Y. Liu, and X. Li, "DiffChaser: Detecting disagreements for deep neural networks," in *28th International Joint Conference on Artificial Intelligence (IJCAI)*, 7 2019, pp. 5772–5778.

[34] Y. Qin, H. Wang, C. Xu, X. Ma, and J. Lu, "SynEva: Evaluating ml programs by mirror program synthesis," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2018, pp. 171–182.

[35] A. Begel and S. L. Graham, "An assessment of a speech-based programming environment," in *Visual Languages and Human-Centric Computing (VL/HCC'06)*. IEEE, 2006, pp. 116–120.

[36] Y.-S. Kim, M. Dontcheva, E. Adar, and J. Hullman, "Vocal shortcuts for creative experts," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–14.