Can Differential Testing Improve Automatic Speech Recognition Systems?

Muhammad Hilmi Asyrofi, Zhou Yang*, Jieke Shi, Chu Wei Quan and David Lo
School of Computing and Information Systems, Singapore Management University
{mhilmia, zyang, jiekeshi, wqchu.2019, davidlo}@smu.edu.sg

***Abstract*—Due to the widespread adoption of Automatic Speech Recognition (ASR) systems in many critical domains, ensuring the quality of recognized transcriptions is of great importance. A recent work, CrossASR++, can automatically uncover many failures in ASR systems by taking advantage of the differential testing technique. It employs a Text-To-Speech (TTS) system to synthesize audios from texts and then reveals failed test cases by feeding them to multiple ASR systems for cross-referencing. However, no prior work tries to utilize the generated test cases to enhance the quality of ASR systems. In this paper, we explore the subsequent improvements brought by leveraging these test cases from two aspects, which we collectively refer to as a novel idea, *evolutionary differential testing*. On the one hand, we fine-tune a target ASR system on the corresponding test cases generated for it. On the other hand, we fine-tune a cross-referenced ASR system inside CrossASR++, with the hope to boost CrossASR++'s performance in uncovering more failed test cases. Our experiment results empirically show that the above methods to leverage the test cases can substantially improve both the target ASR system and CrossASR++ itself. After fine-tuning, the number of failed test cases uncovered decreases by 25.81% and the word error rate of the improved target ASR system drops by 45.81%. Moreover, by evolving just one cross-referenced ASR system, CrossASR++ can find 5.70%, 7.25%, 3.93%, and 1.52% more failed test cases for 4 target ASR systems, respectively.**

***Index Terms*—Automatic Speech Recognition, Test Case Generation, Differential Testing**

I. Introduction

Automatic Speech Recognition (ASR) is a pervasive part of our daily life and has been applied in many critical domains, e.g., voice commands for autonomous driving cars. Enhancing the quality of the automatically recognized transcriptions is essential for people to communicate with intelligent systems seamlessly. An effective way to assure ASR software quality is by performing adequate testing, which requires many test cases in the form of pairs of audios and their corresponding transcriptions. However, manually building such test cases for ASR systems can be time-consuming and resource-intensive, mainly because both collecting inputs (i.e., recording audios) and deciding oracles (i.e., correct transcripts) often require a substantial amount of manual labeling work. To automate the tedious process, researchers proposed some automatic test generation tools for ASR systems, e.g., CrossASR [1] and its successor CrossASR++ [2]. Both tools leverage recent advances in Text-to-Speech (TTS) systems and ASR systems to uncover failed test cases in a purely black-box manner without human supervision. More specifically, they first use a TTS service to transcribe an input text to an audio file and combine them into an audio-text pair. Then, to avoid using the invalid audios synthesized by the TTS system, the generated audio-text pair is fed into multiple ASR systems for cross-referencing: If at least one ASR system correctly recognizes the audio, the audio-text pair is viewed as a valid test case.

Unlike the conventional software systems whose logic is encoded in control and data flows, ASR systems, as other deep learning (DL) programs, encode software behaviors with model parameters and non-linear activation functions. When an ASR system has defects, namely, when an ASR system cannot correctly recognize the audio of a valid test case, developers will collect more data to improve its model. A natural question to ask is how the generated test cases can be leveraged to improve an ASR system. Moreover, as a tool closely relying on ASR systems, CrossASR++ is self-improvable [2]: its ability to uncover failed test cases can be boosted by utilizing better ASR systems for cross-referencing. Thus we are also interested in whether we can improve the performance of CrossASR++ (i.e., in finding more failed test cases) by improving one of the cross-referenced ASR systems with generated test cases.

In this work, we leverage the generated valid test cases along with the running of CrossASR++ to make both the ASR system under test (SUT) and CrossASR++ evolve simultaneously, which we collectively refer to as *evolutionary differential testing*. Specifically, we create a training set by combining all the valid test cases generated so far and fine-tune the SUT on the training set to investigate its consequent improvement. Additionally, we enable CrossASR++ to record test cases identified for cross-referenced ASR systems as well. Since CrossASR++ utilizes the improved ASRs for cross-referencing, it may find more failed test cases for SUTs. To the best of our knowledge, this work is the first exploration of using automatically generated test cases to improve both the ASR system and the test case generation tool. In particular, we empirically evaluate the improvements brought by generated test cases by answering the following two research questions:

RQ1. *Can test cases generated through differential testing be leveraged to improve an ASR system under test?*

RQ2. *Can the generated test cases be leveraged to improve the performance of CrossASR++?*

We design and perform experiments to answer the two questions. First, we consider the evolution of the SUT. We

* Corresponding author.

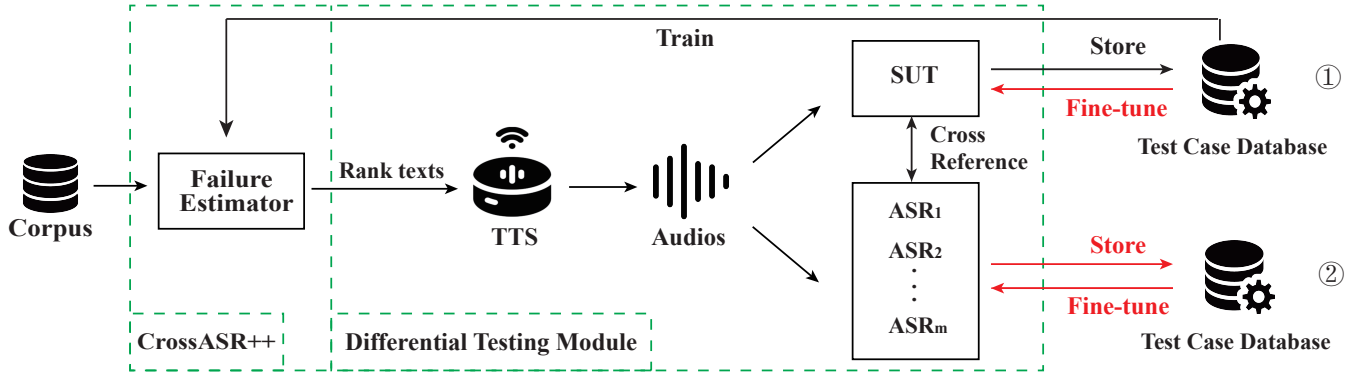

Fig. 1. This figure shows an overview of CrossASR++ and how to utilize its generated test cases to improve a target ASR system and cross-referenced ASR systems. Conceptually, CrossASR++ has two main components: (1) the differential testing module and (2) the failure estimator. The differential testing module takes a piece of text and automatically generates a test case from it. The test cases can be indeterminable, successful, and failed ones. To uncover failed test cases more efficiently, CrossASR++ trains a failure estimator and prioritizes texts that are more likely to become failed test cases. The right side of this figure shows how we leverage generated test cases. We store test cases for both the ASR system under test (SUT) and cross-referenced ASR systems and then fine-tune these ASR systems on their corresponding test cases.

use two metrics to evaluate the performance of the SUT. One is the number of failed test cases uncovered by CrossASR++, and the other is the word error rate (WER) on a test set. The experimental results show that the number of failed test cases uncovered for the improved SUT decreases by 25.81%, and the WER decreases from 7.99% to 4.33% (this corresponds to a *relative* improvement of 45.81%). Then, we consider the evolution of CrossASR++. To measure the improvement of CrossASR++, we compare the numbers of failed test cases uncovered by the original and the evolved CrossASR++. We fine-tune one of the cross-referenced ASR systems and find that such improvement can help CrossASR++ find 5.70%, 7.25%, 3.93%, and 1.52% more failed test cases for the other 4 ASR systems, respectively. The results highlight that the automatically generated test cases can be utilized to enhance both quality of an ASR system and the performance of CrossASR++ itself.

The rest of this paper is organized as follows. Section II briefly describes CrossASR++ and how we utilize the generated test cases. In Section III, we describe our experiments that demonstrate the viability of our new idea. Section IV discusses some related works. Finally, we conclude the paper and present future work in Section V.

II. CrossASR++ and Usage of Generated Tests

This section first presents how CrossASR++ applies differential testing to automatically generate test cases for ASR systems. Then, we explain our idea of evolutionary differential testing that leverages the generated test cases in two aspects: (1) improving the performance of an ASR system under test (SUT) by fine-tuning it, and (2) improving the performance of CrossASR++, i.e., its ability in uncovering failed test cases, by fine-tuning one of the cross-referenced ASRs.

A. CrossASR++

Differential testing is a technique that identifies bugs in software systems by observing whether systems with the same functionality yield different outputs when provided with identical inputs [3]. Differential testing has shown to be effective in revealing failures in deep learning-powered systems, e.g., speech recognition [1], [2], object detection [4], [5] and human activity recognition [6]. We discuss how CrossASR++ performs differential testing to ASR systems as follows.

As shown in Fig. 1, CrossASR++ takes a text corpus as input and tries to generate a test case for the ASR system under test (SUT). Specifically, given a piece of text from the corpus, it first employs a TTS system to synthesize an audio file from the text. The generated audio file is then fed to all ASR systems inside CrossASR++ (SUT and ASR_1 to ASR_m in Fig. 1). The tool cross-references the outputs of these ASR systems and tags the synthesized audio-text pair as a *valid test case* if at least one of the ASR systems can successfully transcribe the audio (i.e., each word in the transcription matches each corresponding word in the input text). A valid test case can be either a *failed test case* or a *successful test case*. If the SUT transcription is different from the input text, it means that CrossASR++ uncovers a failed test case for the SUT. Otherwise, a successful test case is found. We take the audio-text pair as an *indeterminable test case* if none of them can successfully transcribe the audio.

Software testing is usually constrained by time and resource limits. Thus, CrossASR++ selects texts from the corpus strategically to generate test cases more efficiently. CrossASR++ trains a *failure estimator*, which can estimate the probability of a piece of text leading to a failed test case. The failure estimator is a fine-tuned BERT-based classifier with a softmax layer that outputs the probability of an input text being failed, successful and indeterminable test cases. CrossASR++ runs in multiple iterations: in each iteration, it continuously selects texts from the corpus and processes them before a timeout. At the start of each iteration (except the first one), it trains the failure estimator with test cases found so far and estimates the probabilities of texts to be failed test cases. Then, CrossASR++ ranks texts according to their probability in descending order and gives higher priority to the texts with a higher rank. By

doing so, CrossASR++ can uncover more failures in ASR systems before the timeout.

B. Improving ASR Systems

Now, we introduce how the evolutionary differential testing leverages the generated test cases in two aspects.

1) Improving the ASR System Under Test: When software systems are shown to have bugs, programmers or automated program repair tools fix bugs to enhance software quality, e.g., by patching conventional software systems or retraining DL models. In this process, both successful and failed test cases are essential materials to improve the performance of software systems. CrossASR++ runs iteratively, and in each iteration, it generates new test cases for the SUT. We describe how to leverage the generated test cases to evolve the SUT as follows.

We use SUT_0 to denote the initial SUT where CrossASR++ generates test cases for it. CrossASR++ runs in multiple iterations. We use s_i and f_i to represent the successful and failed test cases generated in the i^{th} iteration, respectively. S_i is all the successful test cases generated by CrossASR++ so far (i.e., $S_i = \cup_{n=1}^{i} s_n$). F_i is all the failed test cases generated until the i^{th} iteration except the first iteration. Mathematically speaking, $F_i = \cup_{n=2}^{i} f_n$. We separate f_1 for the evaluation purpose. At the end of the i^{th} iteration, we fine-tune the current model SUT_{i-1} on $S_i \cup F_i$ to achieve a new model SUT_i.

We choose to fine-tune an existing model rather than retraining one, mainly because retraining an accurate ASR system requires a large amount of data, while fine-tuning can quickly improve the model performance on the new dataset by leveraging the information captured in pre-trained models.

2) Improving CrossASR++: Asyrofi et al. [2] have shown that CrossASR++ is self-improvable: its ability in uncovering failed test cases can be boosted by utilizing better ASR systems for cross-referencing. The intuition is that better ASRs systems can recognize more audios correctly to make some indeterminable test cases become valid ones while the original successful and failed test cases remain. This fact motivates us to utilize the generated test cases to improve the cross-referenced ASR systems inside CrossASR++, with the expectation to improve CrossASR++ itself (in terms of its ability to identify more failed test cases for the SUT). We describe the processes as follows.

As an automated testing tool, the original CrossASR++ only stores the test cases generated for the SUT (*test case database* ① in Fig. 1). To make the cross-referenced ASR systems able to evolve as well, we enable CrossASR++ to record both failed and successful test cases for these ASR systems inside it. Recording these test cases introduces not much overhead to our system since we do not need to query ASR systems with audio files again. As shown in Fig. 2, it creates the *test case database* ② that stores the test cases generated for cross-referenced ASR systems in each iteration. Then, it utilizes these test cases to fine-tune the cross-referenced ASR systems in the same way as we evolve an SUT.

III. Experiments

In this section, we describe the dataset and experimental settings used to perform our empirical study. We then present research questions and findings. We end this section by discussing threats to validity.

A. Dataset

We use the Europarl dataset, the same dataset used to evaluate CrossASR++ [2]. The Europarl dataset is a corpus of parallel texts in 11 languages from the proceedings of the European Parliament. We use the English texts that have been collected by CrossASR++'s authors and shared on their project repository[1]. It contains 20,000 texts that are randomly picked from 2,398,750 texts after data cleaning.

B. Experimental Settings

In our experiments, we choose the same TTS system and 5 ASR systems used to evaluate CrossASR++ in [2]. The TTS system is ResponsiveVoice [2]. The 5 ASR systems are DeepSpeech [3], DeepSpeech2 [4], Wav2Letter++ [5], Wav2Vec2 [6], and Wit [7],. We fine-tune one ASR system (DeepSpeech) in this preliminary exploration. We choose to fine-tune DeepSpeech mainly because of the availability of its pre-trained model[8] and the feasibility of fine-tuning the model (e.g., well-maintained documentation and tutorial). For other hyperparameter settings, we use the best settings reported in [2], i.e., 1-hour timeout for each iteration, setting the text batch size to $1,200$, and utilizing Facebook BART-base [7] as the failure estimator.

C. Preliminary Evaluation

We investigate the usefulness of the automatically generated test cases by answering the following research questions.

RQ1. *Can the test cases generated by differential testing be leveraged to improve an ASR system under test?*

Experiment Design. In this research question, we run CrossASR++ using the procedure explained in Section II-B. We fine-tune the ASR system under test (SUT) using generated test cases. We investigate whether the performance of the fine-tuned SUT is better when more iterations are performed. We use DeepSpeech as the SUT. We evaluate the performance of the evolved SUT using 2 metrics, i.e., the number of failed test cases found and the word error rate (WER). For the first evaluation, we run original CrossASR++ and CrossASR++ by evolving the SUT in 5 iterations. We measure the number of failed test cases found in a *static SUT* (non-evolving SUT from CrossASR++) and an *evolved SUT*. We then calculate the reduction of the number of failed test cases after evolution.

[1]https://github.com/soarsmu/CrossASRplus
[2]https://responsivevoice.org/
[3]https://github.com/mozilla/DeepSpeech
[4]https://github.com/PaddlePaddle/DeepSpeech
[5]https://github.com/flashlight/wav2letter
[6]https://huggingface.co/facebook/wav2vec2-base
[7]https://wit.ai/
[8]https://github.com/mozilla/DeepSpeech/releases/tag/v0.9.3

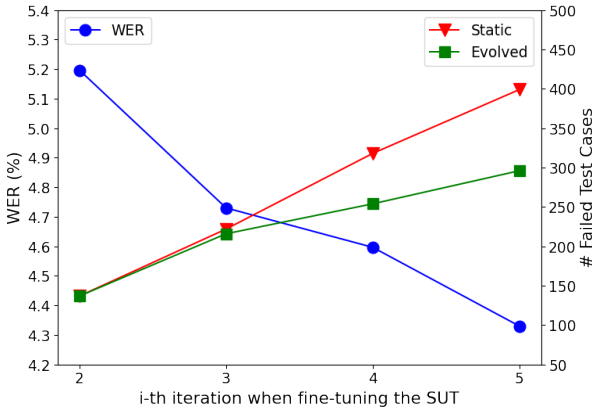

Fig. 2. The performance of the SUT. The blue line with circle markers represents the WER changes of the evolved SUT over time. The green line with square markers represents the number of failed test cases found by the evolved SUT. The red line with triangle markers represents the number of failed test cases found by the static SUT.

For the second evaluation, the WER is the most common metric to evaluate the performance of ASR systems. The word sequence predicted by an ASR system is aligned with a reference transcription. The number of errors from the comparison is calculated as the sum of insertions (*I*), deletions (*D*), and substitutions (*S*) divided by the total words in the reference transcription (*N*). The WER is computed as follows:

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

We measure the WER of the evolved SUT on a hold-out test set. This test set is obtained by taking audio-text pairs from the failed test cases obtained after running the first iteration of CrossASR++. It should be noticed that this test set is not included in the training set of the SUT as explained in Section II-B. We record the change of the WER for each iteration. As a baseline, we also compare the best WER of the evolved SUT with the WER of the static SUT.

Result. Figure 2 shows the performance of the SUT. The horizontal axis represents the number of iterations performed. The left vertical axis represents the WER of the SUT. The right vertical axis represents the number of failed test cases found in the SUT. The red line with triangle markers and the green line with the square markers represent the static and evolved SUT, respectively. In the second iteration, they have the same number of failed test cases because the evolved SUT is not evolved yet. Since the failed test cases found in the first iteration are gathered as test set (see Section II-B), it is still using the original SUT when running the second iteration. By observing the number of failed test cases found after the second iteration, the evolved SUT has fewer failed test cases than the static SUT. The number of failed test cases found decreases from 399 to 296, which is a 25.81% reduction. It points out that the evolved SUT has better performance.

The blue line with circle markers represents the WER changes of the evolved SUT over the iterations. A low WER means the ASR system has low errors when recognizing the audios. The WER of the evolved SUT decreases as the number of iterations increases. It indicates that the evolved SUT successfully learns from the valid test cases. In addition, the WER of the evolved SUT is 4.33%, which surpasses the WER of the static SUT (7.99%). The evolution corresponds to a 45.81% [9] relative improvement of the WER. Overall, it demonstrates that fine-tuning using valid test cases improves the performance of the ASR system under test.

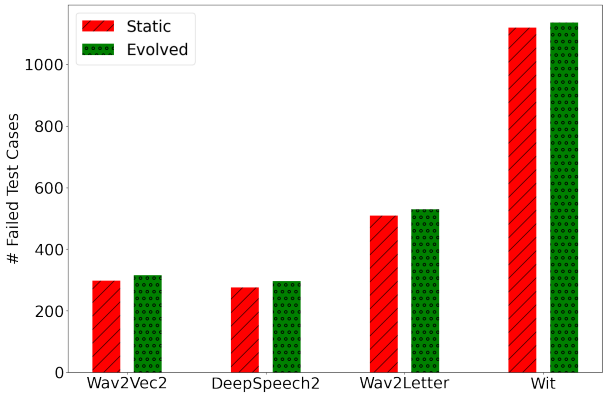

Fig. 3. The number of failed test cases uncovered by the static CrossASR++ and the evolved CrossASR++ in 4 ASR systems. The red dashed bar represents the original CrossASR++. The green dotted bar represents the CrossASR++ by evolving one of the cross-referenced ASR systems.

> **Answers to RQ1**: By fine-tuning the ASR system under test using generated test cases, the number of failed test cases uncovered decreases by 25.81% and the WER decreases by 45.81%.

RQ2. *Can the generated test cases be leveraged to improve the performance of CrossASR++?*

Experiment Design. In this research question, we want to investigate whether fine-tuning one of the cross-referenced ASR systems using generated test cases can boost the performance of CrossASR++ to uncover failed test cases. We compare the number of failed test cases found by original CrossASR++ with the number of failed test cases found by CrossASR++ by evolving one of the cross-referenced ASR systems. For simplification, we call the first as a static CrossASR++ and the second as an evolved CrossASR++. For the evolved CrossASR++, we continuously evolve DeepSpeech and always utilize DeepSpeech as one of the ASR systems used for cross-referencing. We have 4 other ASR systems, i.e., Wav2Vec, DeepSpeech2, Wav2letter, and Wit. We run the static CrossASR++ for 4 times, i.e., by using different ASR systems as SUT for each. We also run the evolved CrossASR++ for 4 times. We record the number of failed test cases found by the static and the evolved CrossASR++ in 4 ASR systems when each of them is used as the SUT.

Result. Figure 3 presents the number of failed test cases found by the static and evolved CrossASR++ in the 4 ASR systems. The red dashed bars and the green dotted bars show the number of failed test cases revealed by the static and evolved CrossASR++, respectively. The horizontal axis shows the ASR

[9] $((7.99 - 4.33)/7.99) \times 100\%$

systems used. The vertical axis shows the number of failed test cases found in each ASR system.

By comparing the number of failed test cases found by the static and evolved CrossASR++ head-to-head, it is clear that the evolved CrossASR++ uncovers more failed test cases than the static one for each ASR system. The numbers of failed test cases found in Wav2Vec2, DeepSpeech2, Wav2letter, and Wit increase by 5.70% (298 to 315), 7.25% (276 to 296), 3.93% (509 to 529), and 1.52% ($1,119$ to $1,136$), respectively.

Answers to RQ2: By fine-tuning one of the cross-referenced ASR systems using generated test cases, the number of failed test cases found by CrossASR++ in Wav2Vec, DeepSpeech2, Wav2letter, and Wit increases by 5.70%, 7.25%, 3.93% and 1.52%, respectively.

D. Threats to Validity

Threat to internal validity concerns factors that may bias our results. The failure estimator's ability in predicting failed test cases can affect answers to RQ2. CrossASR++ may process different texts because the estimator prioritizes different texts in each iteration. To investigate exactly the effect of the ASR evolution for turning indeterminable test cases into valid test cases, we ensure that the texts processed by the evolved CrossASR++ are the same as the ones processed by the static CrossASR++. Thus, we save the texts processed by the static CrossASR++ and process them on the evolved CrossASR++.

Threat to external validity concerns the generalizability of our findings. We only fine-tune one of the ASR systems when evaluating the RQ2. We currently want to focus on analyzing our idea by changing only one ASR system and observing its impact. We choose DeepSpeech as the fine-tuned ASR system because of its reputation, pre-trained model availability, and project documentation clarity as mentioned in Section III-B. We plan to extend our empirical study by evolving other ASR systems in the future.

IV. Related Work

Test Case Generation for ASR systems. DeepCruiser [8] generates test cases using metamorphic transformations guided by coverage criteria. CrossASR++ [2], the successor of CrossASR [1], is the most recent work that uncovers erroneous behaviors in ASR systems using the differential testing. However, they only generate test cases in ASR systems without further action on utilizing these test cases to improve the ASR systems. Our empirical study investigates whether we can leverage the generated test cases to improve the ASR systems.
Repairing (Deep Learning) DL systems. Several works propose repairing techniques when failure cases are found. DeepXplore [4] adopts differential testing to reveal failures cases and repairs DNN models in hand-written digit classification by augmenting failure cases into training data. SENSEI [9] improves the robust generalization of classifiers by re-training the models using generated test cases. DeepRepair [10] proposes a style transfer repairing method to learn and introduce the unknown failure patterns into the training data via data augmentation. Apricot [11] aims to repair DL systems iteratively through a weight-adaptation method. To the best of our knowledge, the previous works repair failure cases in image-related tasks. We are the first that explore the potential usage of generated test cases in repairing ASR systems.

V. Conclusion and Future Work

This paper explores the idea of *evolutionary differential testing*, which uses the test cases generated by differential testing tools to improve both the SUT and the test case generation tool. We empirically show that by fine-tuning the target ASR using generated test cases, the number of failed test cases uncovered decreases by 25.81%, and the WER decreases by 45.81%. In addition, the generated test cases can improve the performance of CrossASR++ in revealing more failed test cases in ASR systems. By fine-tuning one of the cross-referenced ASR systems with generated test cases, the number of failed test cases found by CrossASR++ in Wav2Vec, DeepSpeech2, Wav2letter, and Wit increases by 5.70%, 7.25%, 3.93% and 1.52%, respectively. In the future, we plan to observe and analyze the relative performance of the ASR systems when all of them are evolving. We also plan to investigate better ways to generate effective test cases that can improve the reliability of ASR systems.

Replication Package. The source code for our empirical study is available at https://github.com/soarsmu/ASREvolve.

Acknowledgment. This research was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

References

[1] M. H. Asyrofi, F. Thung, D. Lo, and L. Jiang, "Crossasr: Efficient differential testing of automatic speech recognition via text-to-speech," in *2020 IEEE ICSME*, 2020, pp. 640–650. [Online]. Available: https://doi.org/10.1109/ICSME46990.2020.00066

[2] M. H. Asyrofi, Z. Yang, and D. Lo, "Crossasr++: A modular differential testing framework for automatic speech recognition," in *ESEC/FSE*, Athens, Greece, 2021. [Online]. Available: https://doi.org/10.1145/3468264.3473124

[3] M. A. Gulzar, Y. Zhu, and X. Han, "Perception and practices of differential testing," ser. ICSE-SEIP '19. IEEE Press, 2019, p. 71–80.

[4] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," *Commun. ACM*, vol. 62, no. 11, 2019.

[5] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018.

[6] B. Paulsen, J. Wang, and C. Wang, "Reludiff: Differential verification of deep neural networks," *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pp. 714–726, 2020.

[7] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension." ACL, 2020.

[8] 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.

[9] X. Gao, R. K. Saha, M. R. Prasad, and A. Roychoudhury, "Fuzz testing based data augmentation to improve robustness of deep neural networks," ser. ICSE '20. New York, NY, USA: ACM, 2020.

[10] B. Yu, H. Qi, Q. Guo, F. Juefei-Xu, X. Xie, L. Ma, and J. Zhao, "Deeprepair: Style-guided repairing for dnns in the real-world operational environment," 2020.

[11] H. Zhang and W. Chan, "Apricot: A weight-adaptation approach to fixing deep learning models," in *IEEE/ACM ASE*, 2019.