CrossASR++ : A Modular Differential Testing Framework for Automatic Speech Recognition

Muhammad Hilmi Asyrofi
Singapore Management University
Singapore
mhilmia@smu.edu.sg

Zhou Yang*
Singapore Management University
Singapore
zyang@smu.edu.sg

David Lo
Singapore Management University
Singapore
davidlo@smu.edu.sg

ABSTRACT

Developers need to perform adequate testing to ensure the quality of Automatic Speech Recognition (ASR) systems. However, manually collecting required test cases is tedious and time-consuming. Our recent work proposes CrossASR, a differential testing method for ASR systems. This method first utilizes Text-to-Speech (TTS) to generate audios from texts automatically and then feed these audios into different ASR systems for cross-referencing to uncover failed test cases. It also leverages a failure estimator to find failing test cases more efficiently. Such a method is inherently self-improvable: the performance can increase by leveraging more advanced TTS and ASR systems. So, in this accompanying tool demo paper, we further engineer CrossASR and propose *CrossASR++*, an easy-to-use ASR testing tool that can be conveniently extended to incorporate different TTS and ASR systems, and failure estimators. We also make *CrossASR++* chunk texts from a given corpus dynamically and enable the estimator to work in a more effective and flexible way. We demonstrate that the new features can help *CrossASR++* discover more failed test cases. Using the same TTS and ASR systems, *CrossASR++* can uncover 26.2% more failed test cases for 4 ASRs than the original tool. Moreover, by simply adding one more ASR for cross-referencing, we can increase the number of failed test cases uncovered for each of the 4 ASR systems by 25.07%, 39.63%, 20.95% and 8.17% respectively. We also extend *CrossASR++* with 5 additional failure estimators. Compared to worst estimator, the best one can discover 10.41% more failed test cases within the same amount of time. The demo video for *CrossASR++* can be viewed at https://youtu.be/ddRk-f0QV-g and the source code can be found at https://github.com/soarsmu/CrossASRplus.

CCS CONCEPTS

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

KEYWORDS

Automatic Speech Recognition, Text-to-Speech, Test Case Generation, Cross-Referencing

*Corresponding author.

ESEC/FSE '21, August 23–28, 2021, Athens, Greece

ACM ISBN 978-1-4503-8562-6/21/08…$15.00
https://doi.org/10.1145/3468264.3473124

ACM Reference Format:
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 European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3468264.3473124

1 INTRODUCTION

Automatic Speech Recognition (ASR), an essential technique supporting human-computer interaction, has been widely applied in modern life. Due to its ubiquity, ensuring the quality of ASR systems is important. In software engineering, testing is a common practice to reveal defects, which can then be fixed to improve the quality of software products. Intuitively, a test case for an ASR system is a piece of audio (input) and the corresponding transcription (oracle). However, manually curating these test cases requires significant human effort and time [17]. As a result, researchers have developed automated test generation techniques to reduce the expense for testing ASR systems and helping uncover failures at an early stage.

Such works can be divided into two branches: metamorphic testing and differential testing. The former is built on a basic assumption that adding a subtle disturbance to a piece of audio should not change the recognized transcript produced by an ASR system [1, 5, 6, 15, 20–22]. Work from the other branch, CrossASR [3] that we recently introduced, employs Text-to-Speech (TTS) engines to synthesize test inputs from a text corpus. Then, it performs black-box differential testing on ASR systems by cross-referencing multiple ASRs to detect different transcriptions. If the transcribed text recognized by the ASR system under test does not match the text input to TTS while another ASR's transcribed text does, this input audio is viewed as a failed test case. To increase its efficiency, CrossASR also employs a failure estimator to select texts from the input corpus that are more likely to become failed test cases.

Although these research works and prototypes demonstrate great potential to test ASR systems automatically, the community still lacks an easy-to-use tool that leverages state-of-the-art techniques. However, new TTS and ASR systems are continuously emerging and evolving. For example, wav2letter++ [18], an ASR system used in the CrossASR experiments, is no longer maintained (migrated to another platform) [19] and additional ASR systems, e.g. Wav2Vec2 [4], are newly proposed. CrossASR can inherently improve its ability to uncover failed test cases for a system under test by utilizing more TTS and ASR systems for cross-referencing. These facts motivate us to devote more engineering effort and propose *CrossASR++* in this paper. We modularize the Python implementation of CrossASR. Thus, it can be easily extended to support additional TTS and ASR systems to achieve better performance.

The original CrossASR adopts a *static chunking* strategy: splitting an input corpus into a fixed number of batches. It analyzes only one batch in each iteration (with a specific timeout period). In each iteration, the visibility of the failure estimator is static and limited: it can only access texts in one batch to prioritize them (based on their estimated likelihood to uncover failures). To address this limitation, we design *CrossASR++* with dynamic chunking and flexible visibility. In each iteration, the failure estimator can produce estimates for more texts, and *CrossASR++* can prioritize which texts to be run first (to uncover failures) among a larger and dynamic pool of texts (that changes with each iteration). Our evaluation results show that CrossASR++ outperforms CrossASR by uncovering many more failures given a fixed time budget.

The rest of this paper is organized as follows. Section 2 describes the workflow, main features and a usage example. In Section 3, we evaluate the effectiveness of *CrossASR++* and its features in uncovering failures. Section 4 discusses some related work. Finally, we conclude the paper and present future work in Section 5.

2 *CROSSASR++*

This section discusses the workflow of *CrossASR++* . We also present new features and a usage example of our tool in this section.

2.1 Workflow

lgorithm 1 illustrates the workflow of *CrossASR++* . Given a text corpus, *CrossASR++* processes the corpus gradually in multiple iterations. We first initialize some important variables (Line 1 - 2). At the start of the first iteration, no test cases are processed yet. So the algorithm will skip the body of the `if` statement (Line 5-8) and keep the text batch selected in Line 2. Then, TTS is used to generate an audio file for each piece of selected text (Line 9). In the next step, we use all ASR systems (including the System Under Test (SUT)) to convert the audio files back to texts (Line 11), and then we perform cross-reference (Line 12). For a piece of audio and its corresponding transcription, there are three possible situations: 1) if the SUT can recognize it successfully, i.e. the transcript matches the original text, we take it as a *successful test case*. 2) If the SUT fails to recognize the audio file successfully, but at least one of the other ASR systems succeeds, we take this audio file (and the corresponding input text) as a *failed test case* for the SUT. 3) If all the ASR systems cannot recognize the audio correctly, we call it an *indeterminable test case* because a TTS may generate an invalid audio. After cross-referencing, we store uncovered failed test cases (Line 13) and other test cases (Line 14).

To enable *CrossASR++* to identify more failed test cases given a time budget, we make use of a failure estimator. At the start of each iteration except the first one, we use failed test cases, successful test cases, and indeterminable test cases so far to train a failure estimator (Line 6). This estimator can estimate the probability of a piece of text leading to a failed test cases. We rank the texts to be chosen in the iteration and prioritize texts with a higher ranking to be processed (until the time budget of the iteration has been expended) (Line 7).

By default, the time budget for each iteration is one hour. If the time budget has been expended in an iteration, *CrossASR++* will proceed to the next iteration. So it is possible that *CrossASR++* only processes a part of texts in a batch.

Algorithm 1: *CrossASR++* Workflow

```
   Input: corpus: a list of texts
   Output: failed tests : generated failed test cases
1  failed_tests, other_tests = None, None;
2  texts = getFirstBatch(corpus);
3  while not exceed maximal iteration do
4      ## new text selection;
5      if failed_tests not None then
6          estimator.train(failed_tests, other_tests) ;
7          texts = estimator.select(getNextBatch(corpus));
8      end
9      audios = tts.generateAudio(texts);
10     ## cross-referencing process;
11     transcriptions = asrs.recognizeAudio(audios);
12     f_tests, o_tests = crossReference(transcriptions, texts);
13     failed_tests.append(f_tests);
14     other_tests.append(o_tests);
15 end
16 return failed_tests
```

2.2 Extensibility

We devote more engineering effort to enhancing the extensibility of the original CrossASR. We believe such extensibility can make *CrossASR++* self-improvable: the performance can increase by leveraging more advanced TTS and ASR systems. Theoretically, if no timeout is set for each iteration, adding *CrossASR++* with more ASR systems for cross-referencing will not decrease the number of failed test cases uncovered. The intuition is that newly added ASR systems may turn indeterminable test cases into failed test cases while the original successful and failed test cases remain. As reported in our prior paper [3], TTS systems also differ in their ability to help in finding failed test cases. The main reason is that more advanced TTSes generate fewer invalid audios that are more likely to be indeterminable test cases. As a result, using better TTSes can improve *CrossASR++* . The failure estimator estimate the probability that a text leads to a failed test case. Intuitively, a better failure estimator help *CrossASR++* find more texts that result in failed test cases. The analysis above motivates us to enhance the extensibility of CrossASR by introducing a modular design to CrossASR++.

We discard the implementation of the original CrossASR as it serves more as a prototype to demonstrate the viability of the research idea rather than a tool that others can easily use and expand. We implement all necessary processes presented in Algorithm 1 and pay attention to its extensibility. Extensibility is mainly enabled by modeling a TTS, ASR, and failure estimator with interfaces, i.e. abstract base classes. Users can add a new TTS, a new ASR or a new failure estimator by simply inheriting the base class and implementing necessary methods.

We have 3 base classes, i.e. `ASR`, `TTS`, and `Estimator`. When inheriting from each class, users need to specify a name in the constructor. This name will be associated with a folder for saving the audio files and transcriptions. Thus, each derived class is required to have its own unique name. When inheriting `ASR` class, users must override the `recognizeAudio()` method which takes an audio as input and returns recognized transcription.

The TTS and failure estimator can be added similarly. In `TTS` class, the method `generateAudio()` which converts a text into audio must be overrided by derived classes. In `Estimator` class, methods `fit()` and `predict()` must be overrided by derived classes. These methods are used for training and predicting, respectively.

As default setting of *CrossASR++* , we have incorporated some latest components. The suppported TTSes are Google Translate's TTS [8], ResponsiveVoice [16], Festival [9], and Espeak [7]. The supported ASRs are DeepSpeech [12], DeepSpeech2 [2], Wit [14], and wav2letter++ [18]. *CrossASR++* supports any transformed-based classifier available at HuggingFace [13]. *CrossASR++* can also be easily extended to leverage more advanced tools in the future.

2.3 Dynamic Chunking and Visibility

The original CrossASR adopts a *static chunking* strategy. Before processing the corpus, it splits the corpus into a fixed number of batches and assigns each batch to one iteration. When processing texts in each iteration, CrossASR can only access a limited number of texts assigned in the text batch. If it reaches the time limit, it will discard all the unprocessed texts in the text batch. Also, the estimator can only predict the failure probability of texts in that batch as well. There are two main drawbacks of this *static chunking* strategy. First, the unprocessed texts can also be failed test cases. Thus discarding them will decrease the number of failed test cases. Second, the visibility of the estimator is limited which means that the estimator can only access a small number of texts to prioritize. No matter how effective the estimator is, it can only give a higher rank to a few texts that are likely to be failed test cases.

We design *CrossASR++* with *dynamic chunking* and more flexible visibility. We allow users to adjust the visibility of the failure estimator, and the text batches are no longer split statically before processing. After ranking texts according to their failure probabilities, *CrossASR++* processes the texts starting from the one with the highest failure probability estimate. There is still a timeout (or time budget) for each iteration, but when the timeout is reached, *CrossASR++* appends unprocessed texts to the next iteration rather than simply discards them. In other words, *CrossASR++* chunks texts in each iteration dynamically.

2.4 Tool Usage

2.4.1 Installation. *CrossASR++* can be installed via a simple command: `pip install crossasr`. The components that *CrossASR++* relies on, e.g. TTS, ASR systems and estimators, can also be easily install and incorporated into *CrossASR++* as described in Section 2.2. Please refer to the Github repository for more information.

2.4.2 Configuration and Execution. After adding ASRs, TTSes and estimators, we run *CrossASR++* with some configuration parameters. Users need to set `tts`, `asrs`, `target_asr` and `estimator` parameters to let *CrossASR++* know the TTS to generate audios, ASRs for cross-reference, the ASR under test and the failure estimator to use. Each of these parameters takes a string or a list of strings.

Other important parameters are `num_iteration`, `time_budget`, `text_batch_size`, and `recompute`. *CrossASR++* runs in multiple iterations as specified by `num_iteration`. The timeout for each iteration is limited to `time_budget` measured in second. The number of texts can be processed in each iteration, i.e. the visibility, can be set by specifying `text_batch_size`.

Table 1: Results for *CrossASR++* and the original CrossASR.

Config	# Failed Test Case				Total
	DS	DS2	W2L	Wit	
CrossASR	266	171	391	704	1,532
CrossASR++	319	217	420	978	1,934

Users can adjust these parameters according to their situation. For example, users with a sufficient time can increase `time_budget` and `num_iteration` to generate more failed test cases. In the Github repository, we also provide all the TTS-generated audios we use. Users can set `recompute` as `false` to use them directly.

All of the configuration parameters are saved at `config.json`. We provide an example script `test_asr.py`. Users can execute ASR testing with command `python test_asr.py config.json`.

3 EXPERIMENTS

In this section, we evaluate *CrossASR++* using the Europarl dataset – the same corpus was used to evaluate CrossASR as reported in [3]. After removing duplicates and dropping empty texts, we randomly pick 20,000 texts for our evaluation. To measure how *CrossASR++* performs in finding failed test cases, we answer the following two research questions. For both RQs, we use ResponsiveVoice as TTS, set timeout as 1 hour and run for 5 iterations (with a time budget of 1 hour for each iteration).

RQ1. *How many failed test cases can CrossASR++ find?*

We run *CrossASR++* with four ASR systems under two configurations. The first configuration uses static chunking and the `albert-base-v2` estimator (the setting used by CrossASR), while the second configuration uses dynamic chunking, sets visibility to 1,200, and uses a more advanced estimator (`facebook-bart-base`). These two configurations are helpful for comparing the superiority of *CrossASR++* over CrossASR. Table 1 illustrates the results of running the tool using the two configurations. Each column represents the number of failed test cases uncovered for the system under test (SUT) when we use one ASR as the SUT and the remaining 3 ASRs for cross-referencing. The result shows that for each SUT, *CrossASR++* can help find more failed test cases. In total, *CrossASR++* can find 26.2% more failed test cases than the original tool, which demonstrates significant improvements to the original tool.

RQ2. *To what extent do enhanced features help find more failed test cases?*

We enhance *CrossASR++* with three main features: (1) extensibility to use more ASR, (2) dynamic chunking and flexible visibility of estimator, and (3) extensibility to use advanced estimators. In this RQ, we select the second configuration from RQ1 as the base configuration. By means of an ablation study, we explore how these three features contribute to the performance of CrossASR++; specifically, we modify only one parameter related to one of the features each time and keeping the other parameters untouched.

First, we analyze the effect of extensibility to use more ASRs. Table 2 illustrates the changes in number of failed test cases when adding one more ASR (`Wav2Wec2`) into the system. We can observe that by simply adding one more ASR for cross-referencing can increase the number of failed test cases uncovered for 4 SUTs by 25.07%, 39.63%, 20.95% and 8.17% respectively. The observation aligns with our analysis in Section 2.2.

Table 2: Adding one more ASR to the system.

ASR	Visibility	# Failed Test Case					Total
		DS	DS2	W2L	Wit	W2V	
4	1,200	319	217	420	978	-	1,934
5	1,200	399	303	508	1,058	309	2,577

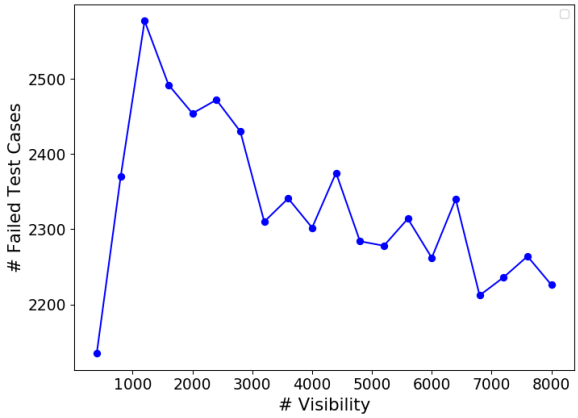

Figure 1: Results when using different visibility settings.

Then, we analyze how dynamic chunking strategy and visibility affect the effectiveness of *CrossASR++*. We use the first configuration from RQ1 and only change static chunking to dynamic chunking. We found that the total number of failed test cases increases from 1,532 to 1,567, emphasizing that dynamic chunking is better. Using the second configuration from RQ1, we also examine how *CrossASR++* performs under different settings of visibility. Figure 1 illustrates the trend of number of failed test cases uncovered when visibility is varied from 400 to 8,000. It can be observed that early in this range (400 to 8,000), using larger visibility can achieve better results than using the original setting, 400. Moreover, there is a peak for visibility's impact, and increasing visibility further reduces the number of failed test cases generated. According to our experiment, 1,200 is an ideal setting for visibility, which can increase the total number of failed test cases by 16.34% compared to setting it to 400.

Lastly, we compare the impacts of using different estimators. We fix other parameters and only replace failure estimators used. We try no estimator and 6 different transformer-based estimators. Table 3 shows the total number of failed test cases when using different estimators. We find that the best result comes from a recently-released model `facebook-bart-base`, which is the default setting of *CrossASR++*. We can observe that estimators can make a difference: the best estimator helps find 10.41% more failed test cases than the worst estimator.

The above analysis shows that all the three enhanced features we incorporate into *CrossASR++* can help boost its performance in finding more failed test cases.

4 RELATED WORK

This section briefly introduces works related to testing ASR systems. We divide current methods to test ASR into two branches, i.e., metamorphic testing and differential testing. Most of works fall into the first branch. Researchers from SE communities [6, 11], follow the conventional software testing principles: define some coverage criteria in ASR systems and generate metamorphic transformations guided by these coverage criteria. Then, they apply transformations to original audios to derive mutants that are expected to have the same recognized transcripts as the original ones. In the AI community, researchers often generate such transformations adversarially by using optimization-based methods [1, 5, 15, 20–22].

Table 3: Results when using different estimators.

Estimator	# Failed Test Cases
No estimator	1,574
bert-base-uncased	2,334
albert-base-v2	2,419
distilbert-base-uncased	2,446
xlnet-base-cased	2,495
roberta-base	2,559
facebook-bart-base	**2,577**

The other branch, differential testing [10], provides the same input to different softwares with the same function and to see whether they produce different outputs. CrossASR [3], our prior work, is the first to apply differential testing to ASR systems. CrossASR employs Text-to-Speech (TTS) engines to synthesize test inputs from texts. Then it performs black-box differential testing on ASR systems by cross-referencing multiple ASRs to detect different transcriptions recognized by ASRs. If the transcribed text recognized by the ASR system under test does not match the text input to a TTS, while another ASR's transcribed text does, this input audio is viewed as a failed test case. To increase its effectiveness to find failed test cases, CrossASR utilizes a failure estimator that selects texts that are more likely to lead to failed test cases.

5 CONCLUSION AND FUTURE WORK

This paper presents *CrossASR++* , an extensible tool that performs black-box differential testing on ASR systems. *CrossASR++* can be conveniently extended to incorporate different TTS and ASR systems, and failure estimators. The extensibility allows *CrossASR++* to flexibly utilizes more tools to boost its performance on uncovering failed test cases. In the default package of *CrossASR++*, we incorporate 4 TTS and 5 ASR systems, as well as 6 failure estimators. Our evaluation results show that *CrossASR++* outperform CrossASR by revealing 26.2% more failed test cases. In addition, we find that the new features introduced by *CrossASR++* help discover more failed test cases. For example, by simply adding one more ASR system for cross-referencing, the number of failed test cases for each of the 4 ASRs under test increases by 25.07%, 39.63%, 20.95% and 8.17% respectively. The number of failed test cases also increases by increasing the visibility until some point. We also find that leveraging a more advanced failure estimator can help *CrossASR++* achieve better performance. We encourage practitioners and researchers to augment *CrossASR++* with more TTS and ASR systems and failure estimators.

ACKNOWLEDGMENTS

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

REFERENCES

[1] H. Abdullah, M. Rahman, W. Garcia, K. Warren, A. Swarnim Yadav, T. Shrimpton, and P. Traynor. 2021. Hear "No Evil", See "Kenansville": Efficient and Transferable Black-Box Attacks on Speech Recognition and Voice Identification Systems. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 712–729. https://doi.org/10.1109/SP40001.2021.00009

[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. http://proceedings.mlr.press/v48/amodei16.html

[3] M. H. Asyrofi, F. Thung, D. Lo, and L. 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)*. 640–650. https://doi.org/10.1109/ICSME46990.2020.00066

[4] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. arXiv:2006.11477 [cs.CL]

[5] N. Carlini and D. Wagner. 2018. Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. In *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–7. https://doi.org/10.1109/SPW.2018.00009

[6] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Jianjun Zhao, and Yang Liu. 2018. DeepCruiser: Automated Guided Testing for Stateful Deep Learning Systems. arXiv:1812.05339 [cs.SE]

[7] Jonathan Duddington. [n.d.]. eSpeak TTS. http://espeak.sourceforge.net. Accessed: 2021-04-30.

[8] Pierre Nicolas Durette. [n.d.]. Google Translate's Text-to-Speech. https://pypi.org/project/gTTS/. Accessed: 2021-04-30.

[9] The Centre for Speech Technology Research. [n.d.]. The Festival Speech Synthesis System. https://www.cstr.ed.ac.uk/projects/festival/. Accessed: 2021-04-30.

[10] Muhammad Ali Gulzar, Yongkang Zhu, and Xiaofeng Han. 2019. Perception and Practices of Differential Testing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice* (Montreal, Quebec, Canada) *(ICSE-SEIP '19)*. IEEE Press, 71–80. https://doi.org/10.1109/ICSE-SEIP.2019.00016

[11] Jianmin Guo, Yue Zhao, Quan Zhang, and Yu Jiang. 2021. RNN-Test: Towards Adversarial Testing for Recurrent Neural Network Systems. arXiv:1911.06155 [cs.CL]

[12] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2014. Deep Speech: Scaling up end-to-end speech recognition. arXiv:1412.5567 [cs.CL]

[13] HuggingFace. [n.d.]. HuggingFace. https://huggingface.co. Accessed: 2021-04-30.

[14] Wit.ai Inc. [n.d.]. Wit. https://wit.ai. Accessed: 2021-04-30.

[15] Shreya Khare, Rahul Aralikatte, and Senthil Mani. 2019. Adversarial Black-Box Attacks on Automatic Speech Recognition Systems using Multi-Objective Evolutionary Optimization. arXiv:1811.01312 [cs.CR]

[16] Tino Khong. [n.d.]. ResponsiveVoice TTS. https://pypi.org/project/rvtts/. Accessed: 2021-04-30.

[17] D. S. Pallet, W. M. Fisher, and J. G. Fiscus. 1990. Tools for the analysis of benchmark speech recognition tests. In *International Conference on Acoustics, Speech, and Signal Processing*. 97–100 vol.1. https://doi.org/10.1109/ICASSP.1990.115546

[18] Vineel Pratap, Awni Hannun, Qiantong Xu, Jeff Cai, Jacob Kahn, Gabriel Synnaeve, Vitaliy Liptchinsky, and Ronan Collobert. 2019. Wav2Letter++: A Fast Open-source Speech Recognition System. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 6460–6464. https://doi.org/10.1109/ICASSP.2019.8683535

[19] Vineel Pratap, Awni Hannun, Qiantong Xu, Jeff Cai, Jacob Kahn, Gabriel Synnaeve, Vitaliy Liptchinsky, and Ronan Collobert. 2019. Wav2Letter++: A Fast Open-source Speech Recognition System. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (May 2019). https://doi.org/10.1109/icassp.2019.8683535

[20] Yao Qin, Nicholas Carlini, Ian J. Goodfellow, Garrison W. Cottrell, and Colin Raffel. 2019. Imperceptible, Robust, and Targeted Adversarial Examples for Automatic Speech Recognition. In *ICML*. http://proceedings.mlr.press/v97/qin19a/qin19a.pdf

[21] Lea Schönherr, Katharina Kohls, Steffen Zeiler, Thorsten Holz, and Dorothea Kolossa. 2018. Adversarial Attacks Against Automatic Speech Recognition Systems via Psychoacoustic Hiding. arXiv:1808.05665 [cs.CR]

[22] Rohan Taori, Amog Kamsetty, Brenton Chu, and Nikita Vemuri. 2019. Targeted Adversarial Examples for Black Box Audio Systems. In *2019 IEEE Security and Privacy Workshops (SPW)*. 15–20. https://doi.org/10.1109/SPW.2019.00016