USENIX Security '25 Artifact Appendix: SafeSpeech: Robust and Universal Voice Protection Against Malicious Speech Synthesis

Zhisheng Zhang[1], Derui Wang[2] ✉, Qianyi Yang[1], Pengyang Huang[1], Junhan Pu[1], Yuxin Cao[3], Kai Ye[4], Jie Hao[1] ✉, and Yixian Yang[1]

[1]*Beijing University of Posts and Telecommunications* [2]*CSIRO's Data61*

[3]*National University of Singapore* [4]*The University of Hong Kong*

A Artifact Appendix

A.1 Abstract

In our artifact, we provide the source code with a detailed description of SafeSpeech in the public communities. We have implemented the algorithmic design of SafeSpeech and open-sourced our code at https://github.com/wxzyd123/SafeSpeech on GitHub and https://zenodo.org/records/15118529 on Zenodo with the final version. In this artifact, we introduce step-by-step instructions on reproducing the experimental results of the main experiments in the paper, including dependence description, environment installation, data preparation, and running commands with expected outputs.

A.2 Description & Requirements

We provide minimum hardware and software requirements for reproducing our experiments.
Hardware Requirements: CPU: a device with 100GB; GPU: NVIDIA A800 with 80GB memory or other GPU devices with the same memory.
Software Requirements: Operating system: Linux (Ubuntu 20.04); Python 3.8; PyTorch 2.0.0; CUDA 12.4. The required Python libraries are provided in `requirements.txt`.

A.2.1 Security, privacy, and ethical concerns

There are no security, privacy, or ethical concerns in this artifact.

A.2.2 How to access

The source code for SafeSpeech is available on GitHub. You can download it directly from the repository page or use the Git tool to clone it. The repository includes the SafeSpeech implementation and the LibriTTS dataset.

✉ Corresponding authors: `derek.wang@data61.csiro.au`, `haojie@bupt.edu.cn`.

A.2.3 Hardware dependencies

We recommend a device equipped with the Ubuntu operating system version 20.04 for hardware configuration. In our tests, the CPU memory is 100 GB. We recommend using a device with more than 80GB for the GPU, such as the NVIDIA A800 described in our paper.

A.2.4 Software dependencies

For software configuration, our experiments are conducted in an environment with Python 3.8, PyTorch 2.0.0, and CUDA 12.4. The SafeSpeech implementation is built on the PyTorch framework and includes additional Python libraries, which are specified in the `requirements.txt` file.

A.2.5 Benchmarks

In this paper, we evaluate the effectiveness, transferability, and robustness of SafeSpeech primarily on speakers from the LibriTTS [2] dataset who are most similar to the pre-trained speakers on BERT-VITS2 [1] model for perturbation generation and fine-tuning. The original data is provided in the `data/LibriTTS` folder within the source code, requiring no further modifications.

A.3 Set-up

A.3.1 Installation

Our experiments are conducted in a Conda environment. First, install the CUDA toolkit (version 11.8 or higher) to ensure proper dependency setup. Then, you can install the Conda environment from the website https://www.anaconda.com/download.

For environment creation, we provide the corresponding instructions in the "Setup" section of the `README`.

(1): `conda create -name safespeech python=3.8`: This command creates a conda environment named "safespeech" with Python version 3.8.

(2): `conda activate safespeech`: This switches to the "safespeech" conda environment.

(3): `pip install -r requirements.txt`: This command installs the necessary Python dependencies required to run the experiment.

(4): `sudo apt install ffmpeg`: This command installs the "ffmpeg" software.

A.3.2 Basic Test

To check whether PyTorch has been successfully installed, you can try adding the following lines to a .py file with "import torch, touchaudio" and "print(torch.cuda.is_available())". Then, run the script. If the command line output is "True", it means the basic test has passed.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): SafeSpeech achieves the MCD value of 14.771 after fine-tuning on LibriTTS dataset and the BERT-VITS2 model, reflecting the higher protective performance than baselines in Table 1;
(C2): The evaluation values of WER and MCD are 99.610% and 0.204 which are better than baselines.

A.4.2 Experiments

(E1): *[Pre-trained Models][1 human-minute + 10 compute-minutes]* For downloading the pre-trained models of the BERT-VITS2, you can run this command:
Command: python download_models.py;
Output: The downloaded checkpoints.

(E2): *[Protection][1 human-minute + 25 compute-minutes]* Since LibriTTS is already available in the GitHub repository, you can start the experimental evaluation once the pre-trained models are prepared. The evaluation process is mainly divided into protection, fine-tuning, and evaluation.
First, you need to generate the BERT file for each audio by DeBERTa model:
Command: python bert_gen.py –dataset LibriTTS –mode clean;
Output: BERT files.
The generated BERT files can be found in the `data/LibriTTS/protected/clean` folder. After obtaining BERT files, the protected audio can be obtained by:
Command: python protect.py –dataset LibriTTS –batch-size 27 –mode SPEC;
Output: protective perturbations.

(E3): *[Saving][1 human-minute + 2 compute-minutes]* The generated perturbations have been saved to `checkpoints/LibriTTS/noises` folder; then you can add the perturbations to the original audio to get protected data:
Command: python save_audio.py –mode SPEC –batch-size 27;
Output: protected audio.
The saved audio files can be found in the folder `data/LibriTTS/protected/SPEC`.

(E4): *[Fine-tuning][1 human-minute + 20 compute-minutes]* Following the aforementioned commands, the dataset has been protected. You can utilize these audio files for fine-tuning. Also, the BERT files need to be generated:
Command: python bert_gen.py –dataset LibriTTS –mode SPEC;
Output: BERT files.
The generated BERT files of protected audio can be found at `data/LibriTTS/protected/SPEC`. Then, you can fine-tune BERT-VITS2 by:
Command: python train.py –mode SPEC –batch-size 64;
Output: checkpoint of fine-tuning.
The checkpoint of the generator after fine-tuning has been saved to `checkpoints/LibriTTS`.

(E5): *[Evaluation][1 human-minute + 10 compute-minutes]* The last step of the workflow is an evaluation after fine-tuning by this command:
Command: python evaluate.py –mode SPEC;
Output: the synthesized results of three metrics.
The synthesized audio can be found in the folder `evaluation/LibriTTS/SPEC` and the output information is:
Mode SPEC, MCD: XXX
Mode SPEC: GT WER is 0.000000, Syn WER is XXX
Mode SPEC on LibriTTS, SIM XXX, ASR XXX.
These three lines represent the evaluation metrics MCD, WER (*i.e., Syn WER*), and SIM, respectively. Your results may not be the same as ours (MCD 14.771, WER 99.610, and SIM 0.204), but they should be higher than baselines, *i.e.*, the values in Table 1, D_1 dataset, under the BERT-VITS2 model, excluding the SPEC row.

Additionally, to facilitate quick validation of SafeSpeech, we provide a Colab version via Jupyter Notebook that includes environment setup and experimental replication steps.

A.5 Notes on Reusability

In this artifact, we provide detailed instructions on how to reproduce SafeSpeech on a new device and environment. All the pre-trained models and datasets are open-source, and we have introduced how to download and protect them by SafeSpeech.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.

References

[1] Bert-vits2. https://github.com/fishaudio/Bert-VITS2, 2024.

[2] H. Zen, V. Dang, R. Clark, Y. Zhang, R. J. Weiss, Y. Jia, Z. Chen, and Y. Wu. Libritts: A corpus derived from librispeech for text-to-speech. *arXiv*, 2019.