USENIX Security '25 Artifact Appendix: Sound of Interference: Electromagnetic Eavesdropping Attack on Digital Microphones Using Pulse Density Modulation

Arifu Onishi*
The University of Electro-Communications

S. Hrushikesh Bhupathiraju*
University of Florida

Rishikesh Bhatt
University of Florida

Sara Rampazzi
University of Florida

Takeshi Sugawara
The University of Electro-Communications

A Artifact Appendix

A.1 Abstract

As described in the paper, we discovered a new side-channel attack on pulse density modulation (PDM) digital MEMS microphones which are used in common laptops and IoT devices. In the paper, discover that each harmonic of these digital pulses retains acoustic information, allowing the original audio to be retrieved through simple FM demodulation using standard radio receivers, even in behind-the-wall scenarios. In this artifact appendix, we provide spoken digits and spoken sentence audio data reconstructed by the attack across various devices using a 14-element Yagi antenna, a magnetic loop antenna, and a cheap antenna made from copper foil. The artifacts also include the scripts to fine-tune and evaluate the HuBERT transformer model for digit and speaker recognition and evaluate the data with HuBERT, OpenAI, and Microsoft speech transcription models required to reproduce the results described in the paper (more specifically, the outcome of the analysis depicted in Tables 4, 5, 6, 7, and 8 of the paper).

A.2 Description & Requirements

The artifact contains scripts (and models) to fine-tune Transformer models for speech recognition, API scripts for inferring and evaluating speech transcription.

A.2.1 Security, privacy, and ethical concerns

Our artifact is non-destructive, and there is no security or privacy risk to the evaluators when the training, fine-tuning, and inference models.

A.2.2 How to access

Our research artifacts, including the speech transcription results and training log files for digit and speaker classification models, along with the corresponding recorded audio files for the tested devices and the scripts for training and fine-tuning the pipeline we used for our speech recognition and transcription models are available at: https://zenodo.org/records/14736347.

We include a README file in the repository to guide the reviewers through the setup of the environment, and the information regarding the provided data corresponding to the experiment in the paper.

A.2.3 Hardware dependencies

We build our recognition models by fine-tuning HuBERT transformer models on our dataset. To effectively fine-tune HuBERT models, you typically need a GPU with at least 16GB of VRAM. Our model was fine-tuned on a single Nvidia A100 GPU (40 GB), with a fine-tuning duration of approximately 45 minutes.

A.2.4 Software dependencies

There is no dependency on the Operating System used by the user. To run the inference codes, the user needs access to Python 3 (above 3.8), conda environment with the following Python packages: pytorch==2.4.0, torchvision==0.19.0, torchaudio==2.4.0, pytorch-cuda=11.8, pandas, scipy, tqdm, matplotlib, seaborn, scikit-learn, transformers, openpyxl, PyWavelets, librosa, datasets, openai, and azure-cognitiveservices-speech.

Detailed instructions to setup the conda environment and the python libraries are included in the README file available at: https://zenodo.org/records/14736347. Note that running the OpenAI and Microsoft APIs for speech transcription models requires paid subscriptions and the API keys. The HuBERT transcription model on the other hand can be implemented and evaluated using the provided scripts.

A.2.5 Benchmarks

All the data required for the artifacts are available at the provided link.

A.3 Set-up

A.3.1 Installation

Run the following commands to configure the conda environment.

- conda create -y –name SoI python==3.8.19
- conda activate SoI
- conda install pytorch==2.4.0 torchvision==0.19.0 torchaudio==2.4.0 pytorch-cuda=11.8 -c pytorch -c nvidia
- pip install pandas scipy tqdm matplotlib seaborn scikit-learn transformers openpyxl PyWavelets librosa
- pip install datasets
- pip install openai
- pip install azure-cognitiveservices-speech
- pip install jiwer

The test the performance of the attack, download the reconstructed audio from `https://zenodo.org/records/14736347` and run the python scripts included in the built conda environment.

A.3.2 Basic Test

To evaluate the Word Error Rates (WER) transcription results from the CSV files provided, run *Evaluation_Results/Transcription/evaluation_wer.py* by changing the CSV file path to the evaluation experiment. To test the fine-tuning on digit recognition by the running the following command: *python3 HuBERT_emispeech_pre_trained_digit_Classifier.py –json_file sample.json* by changing the path in sample.json file to the following:
'/Feasibility/FSDD'.

To run the speaker recognition, replace 'digit' in the script name with 'speaker' and run the following command: *python3 HuBERT_emispeech_pre_trained_speaker_Classifier.py –json_file sample.json*

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The results shown in Tables 4, 5, 6, 7, and 8 for digit and speaker recognition can be achieved by fine-tuning the HuBERT model on the FSDD dataset reconstructed from the attack (provided in this artifact).

(C2): The speech transcription results shown in Table 4, 5, and 8 can be reproduced by running the transcription_hubert.py, transcription_OpenAI.py, and transcription_mirosoft.py scripts on the corresponding data as explained in A.4.2.

A.4.2 Experiments

(E1): *[Generality across off-the-shelf PDM Microhpones] [10 human-minutes + 5 compute-hour]: This experiment reproduces the results shown in Table 4*

How to: In this experiment we run the FSDD and Harvard Sentences data collected from different PDM microphones to achieve speech recognition and transcription respectively.

Preparation: To get the fine-tune digit and speaker recognition results, edit the path for the dataset in the sample.json file with the following directories

- '/PDMs/Knowles/FSDD'
- '/PDMs/STM/FSDD'
- '/PDMs/TDK-41350/FSDD'
- '/PDMs/TDK-T3902/FSDD'
- '/PDMs/VM3000/FSDD'

To run speech transcription using HuBERT, OpenAI, and Microsoft Speech Studio, change the directory path in *transcription_hubert.py, transcription_OpenAI.py, and transcription_mirosoft.py* scripts with the below paths iteratively.

- '/PDMs/Knowles/Harvard_Sentences'
- '/PDMs/STM/Harvard_Sentences'
- '/PDMs/TDK-41350/Harvard_Sentences'
- '/PDMs/TDK-T3902/Harvard_Sentences'
- '/PDMs/VM3000/Harvard_Sentences'

Execution: Run the following script to fine-tune the digit and speaker recognition models: *python3 HuBERT_emispeech_pre_trained_digit_Classifier.py –json_file sample.json*. (Replace 'digit' in the file name with 'speaker' to run fine-tuning on speaker recognition) Run the following commands to run the speech transcription models

- *python transcription_hubert.py*
- *python transcription_OpenAI.py*
- *python transcription_microsoft.py*

Results: The results for the fine-tuning models will be printed on the console. The results can also be saved into a log file to check the final model performance. Note that the classification accuracy can vary slightly due to randomized train-val split. The results from the transcription scripts will be saved to csv files with the audio file name and the corresponding transcription results. *Evaluation_Results/Transcription/evaluation_wer.py* can be used to extract the Word Error Rate (WER) from transcription results in CSV files.

(E2): *[Behind the Wall Scenarios] [10 human-minutes + 5 compute-hour]: This experiment reproduces the results shown in Table 5*

How to: In this experiment we run the FSDD and Harvard Sentences data collected from behind-the-wall scenarios.

Preparation: To get the fine-tune digit and speaker recognition results, edit the path for the dataset in the sample.json file with the following directories

- '/Behind_the_wall/Audio_Microphone/FSDD'
- '/Behind_the_wall/Loop/15cm/FSDD'
- '/Behind_the_wall/Loop/20cm/FSDD'
- '/Behind_the_wall/Loop/25cm/FSDD'
- '/Behind_the_wall/Copper/15cm/FSDD'
- '/Behind_the_wall/Copper/20cm/FSDD'
- '/Behind_the_wall/Copper/25cm/FSDD'

To run speech transcription using HuBERT, OpenAI, and Microsoft Speech Studio, change the directory path in *transcription_hubert.py, transcription_OpenAI.py, and transcription_mirosoft.py* scripts with the below paths iteratively.

- '/Behind_the_wall/Audio_Microphone/Harvard_Sentences'
- '/Behind_the_wall/Loop/15cm/Harvard_Sentences'
- '/Behind_the_wall/Loop/20cm/Harvard_Sentences'
- '/Behind_the_wall/Loop/25cm/Harvard_Sentences'
- '/Behind_the_wall/Copper/15cm/Harvard_Sentences'
- '/Behind_the_wall/Copper/20cm/Harvard_Sentences'
- '/Behind_the_wall/Copper/25cm/Harvard_Sentences'

Execution: Run the following script to fine-tune the digit and speaker recognition models: *python3 HuBERT_emispeech_pre_trained_digit_Classifier.py – json_file sample.json*. (Replace 'digit' in the file name with 'speaker' to run fine-tuning on speaker recognition) Run the following commands to run the speech transcription models

- *python transcription_hubert.py*
- *python transcription_OpenAI.py*
- *python transcription_microsoft.py*

Results: The results for the fine-tuning models will be printed on the console. The results can also be saved into a log file to check the final model performance. Note that the classification accuracy can vary slightly due to randomized train-val split. The results from the transcription scripts will be saved to csv files with the audio file name and the corresponding transcription results. *Evaluation_Results/Transcription/evaluation_wer.py* can be used to extract the Word Error Rate (WER) from transcription results in CSV files.

(E3): *[Long Distance Experiments] [10 human-minutes + 5 compute-hour]: This experiment reproduces the results shown in Table 6*

How to: In this experiment we run the FSDD data collected from long distances in behind-the-wall scenarios.

Preparation: To get the fine-tune digit and speaker recognition results, edit the path for the dataset in the sample.json file with the following directories

- '/Long_Distance/1m'
- '/Long_Distance/2m'
- '/Long_Distance/3m'
- '/Long_Distance/4m'

Execution: Run the following script to fine-tune the digit and speaker recognition models: *python3 HuBERT_emispeech_pre_trained_digit_Classifier.py – json_file sample.json*. (Replace 'digit' in the file name with 'speaker' to run fine-tuning on speaker recognition)

Results: The results for the fine-tuning models will be printed on the console. The results can also be saved into a log file to check the final model performance. Note that the classification accuracy can vary slightly due to randomized train-val split.

(E4): *[Room Scenario Evaluation] [10 human-minutes + 5 compute-hour]: This experiment reproduces the results shown in Table 7*

How to: In this experiment we run the FSDD data collected from long distances in behind-the-wall scenarios.

Preparation: To get the fine-tune digit and speaker recognition results, edit the path for the dataset in the sample.json file with the following directories

- '/Long_Distance/1m'
- '/Room_Scenarios/Concrete_wall'
- '/Room_Scenarios/Oriented'
- '/Room_Scenarios/Through_Monitor'

Execution: Run the following script to fine-tune the digit and speaker recognition models: *python3 HuBERT_emispeech_pre_trained_digit_Classifier.py – json_file sample.json*. (Replace 'digit' in the file name with 'speaker' to run fine-tuning on speaker recognition)

Results: The results for the fine-tuning models will be printed on the console. The results can also be saved into a log file to check the final model performance. Note that the classification accuracy can vary slightly due to randomized train-val split.

(E5): *[Automated Fine-tuning Evaluation] [10 human-minutes + 20 compute-hour]: This experiment will automatically reproduce all the fine-tuning results discussed until now.*

How to: In this experiment we automatically run the FSDD fine-tuning on all the data.

Preparation: To get the fine-tuned digit and speaker recognition results, prepare the data as follows:

- 'unzip Evaluation_Scripts.zip'
- 'cd Evaluation_Scripts'
- 'chmod +x evaluate.sh'

Execution: Run the following command to fine-tune and test the digit and speaker recognition models for all the data: *'./evaluate.sh'*.

Results: Upon running evaluate.sh, all compressed FSDD data directories are automatically extracted into the same directory as the script. Following extraction, the digit and speaker recognition pipelines are executed sequentially. Each pipeline generates a corresponding results directory—suffixed with either _digit or _speaker—within the same path. These result folders contain all relevant artifacts, including fine-tuning logs, training and testing metrics, confusion matrices, and model checkpoints.

(E6): *[Automated Transcription Evaluation] [10 human-minutes + 2 compute-hour]: This experiment will automatically reproduce all the transcription results discussed until now.*

How to: In this experiment we automatically run the speech transcription on all the data.

Preparation: To get the speech transcription, prepare the data as follows:

- 'cd Evaluation_Scripts'
- 'chmod +x evaluate_transcription.sh'

Execution: Run the following command to get transcription for all the Harvard_Sentences data: *'./evaluate_transcription.sh'*.

Results: Upon running evaluate_transcription.sh, all relevant audio directories containing Harvard Sentences are processed from the same directory as the script. The transcription evaluation is performed sequentially using HuBERT (local), OpenAI Whisper API, and Microsoft Azure Speech API. Each backend generates a corresponding CSV output, prefixed with the backend name (e.g., hubert_, openai_, microsoft_) and suffixed with the name of the data directory. All transcription results are stored in a shared transcription_results folder alongside the script.

A.5 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/.