6.857 Final Project: Looming Over Zoom

Aaditya Singh
aaditya@mit.edu

Lauren Oh
laurenoh@mit.edu

William Luo
wqcluo@mit.edu

May 13, 2020

Abstract

Covert keylogging has been an active area over the past few years, but in light of the recent events with COVID-19, eavesdropping over video-conferencing apps is now even more of a concern than ever before. In order to investigate the security vulnerabilities present in video-conferencing apps, we attempt to develop a system for covert keylogging over Zoom, one of the most popular video-conferencing platforms during the 2020 pandemic. We utilize peakfinding algorithms for keystroke detection and a convolutional neural network architecture for keystroke classification, and achieve above-chance top-1 accuracies of around 10% on held-out data. Although above chance, such a low accuracy is likely not sufficient to recover full typed passwords from audio recordings, at least demonstrating the difficulties of extracting key strokes from Zoom audio data. We hope to investigate additional architectures like RNNs or Transformers, which could additionally incorporate time between keystrokes, for future work.

Keywords: security; eavesdropping; keylogging; machine learning.

1 Introduction

Eavesdropping has been an active area of research for almost two decades. Modern security software knows to look for viruses and bugs that are embedded in the applications utilized by computer users, but can fail to account for audio eavesdropping later analyzed to pick up passwords. Acoustic eavesdropping over video-conferencing platforms is a relatively new field, given the increase in video-conferencing usage over the last few years. In the modern day, with the quick spike in video conferencing software usage seen in 2020 alone, acoustic eavesdropping becomes a greater potential threat and vulnerability. Compared to eavesdropping physically using external devices (e.g. mobile device recordings), video conferences and their recordings are a much more vulnerable way to exploit keystroke recordings to extract passwords. Our project aims to combine previous research in the general field of keylogging with its implications on video-conferencing apps, specifically Zoom.

2 Background

2.1 Keylogging

Keystroke logging, commonly referred to as keylogging or keyboard capturing, occurs when the keys struck on a keyboard are recorded or logged. Typically, keylogging is used

nefariously to monitor, record, and retreive the keystroke data of an unaware user. Traditionally, malicious keylogging has been used to eavesdrop passwords and other confidential information.

A wide variety of keylogging attacks work, ranging from using Wifi transmitters to detect keystrokes [1] to using inter-key timing [2] to detecting keys from audio [3] [4]. In our work, we primarily concern ourselves with this last style of attack, as we feel it is of the most import in the evolving situation we find ourselves in - that of social distancing and video calls.

2.2 Previous Work

Most previous work on audio keylogging has mainly considered it in four steps:

1. Keystroke Detection: Isolating keystrokes from continuous audio of typing
2. Window (and feature) extraction: Select windows and extract auditory features around keystroke times from step 1
3. Keystroke classification: Classify the keystroke for each window
4. Error correction (optional): Fix errors that can be identified by considering keys before/after the given key. Error correction is typically used for keylogging natural language (as language models can be used for error correction), but is not as applicable in passwords, where a language model is not of as much use.

The first approach to audio-based keylogging back in 2004 actually only considered steps 2 and 3, so it did not constitute a full attack. Specifically, Asonov and Agrawal [3] found that using Fourier features from the keypress audio lead to good classification performance (97% accuracy). However, their approach was limited in its applicability to continuous streams of typing, did not account for noise This approach to audio-based keylogging was first outlined by Asonov and Agrawal in 2004 [3], who attempted to use the audio of a single keystroke to identify the keystroke. As a result, they only really addressed steps 2 and 3 of the above pipeline, and even so did this for a small set of keys. Their main contribution to the field, thus, was that they identified what causes differential sounds for keystrokes; namely, when a keyboard strikes the underlying "keyboard plate", a distinct sound is made based on where on the plate the strike was. This information can be used to identify the keystroke!

Zhuang et al. 2009 built on top of [3] by encompassing the full 4 steps detailed above [4]. Furthermore, their attack has the benefit of being unsupervised, as the final error-correction step involves unsupervised clustering of predicted characters by using a Hidden Markov Model (HMM) language model. However, their treatment of the first two steps, thresholding for keystrokes and picking 100ms windows, is not applicable in general scenarios. For example. in the data we collected, we found that roughly a quarter of keystrokes have less than 100ms between them. Furthermore, thresholding assumes

relatively constant peak amplitudes which is not the case for most typers. The main contribution of their paper (beyond the HMM-based error correction) was the use of cepstral coefficients (as opposed to raw frequencies from the Fourier transform). Cepstral coefficients are often used in speech processing, and are basically a binned version of the FFT.

After these two seminal papers, the field has grown, with some authors even considering a video conferencing scenario over Skype [5]. However, many of these approaches are lacking in their generalizability. A recent paper [6] which we based some our methods off of aims to generalize better. Specifically, they use modern deep learning techniques like Convolutional Neural Networks (CNNs) for keystroke detection and classification, combined with error-correcting Recurrent Neural Networks (a more "neural" version of HMMs). Although their approach reports a much lower top-1 accuracy of 41%, this accuracy is much more akin to what an attacker would actually have to deal with. The downside to their approach is that it involves many modalities of sensor data (collected from a smart phone on the same table as the targeted laptop) and is thus not easily extensible to a video conferencing scenario. Furthermore, use of an error-correcting language model can lead to difficulties in applying the technique to passwords.

As will be seen in Section 3, our methods are motivated in large part by the literature we read. We have introduced modifications where we feel the literature was either lacking, not applicable, or limiting. Before we detail our approach, however, we introduce a unique challenge we faced that was not considered in the literature.

2.3 Unique challenge: Zoom Audio Filtering

In a 2018 update, Zoom released an upgraded version of their audio processing algorithms to detect and suppress background noise. Specifically, Zoom detects keystroke background noises and scales them down, all while automatically adjusting the rest of the speaker's noises (i.e. their speech) to an optimal volume.

This particular audio feature was one our group was not aware of before we began our project. We discovered its effects when our Zoom keystroke recordings, which were done in a quiet environment, resulted in two or three loud keystrokes followed by scaled down soft silence and no discernible keystroke sounds. While the keystrokes are not completely silent and still noticeable in the waveform graphs of these audio recordings, they are not distinguishable to the human ear.

Zoom's suppression and scaling proved to make our learning more difficult since any notable volume differences between keys were scaled down to appear less relevant. Furthermore, scaling the waveforms down decreases the signal-to-noise ratio of our inputs, as other soft noise peaks (i.e. from background shuffle noises or vibrations from nearby cellphones) appear at similar volumes to keystroke peaks.

3 Our approach

Our approach follows the first three steps mentioned at the start of Section 2.2 with a focus on generalizing to different typing speeds and dealing with the noisier environment present in Zoom.

3.1 Keystroke detection

For keystroke detection, we opted to do something in between the simple threshold-based system from [4] and the CNN-based detection used in [6]. We used a simple peakfinder with our own modifications. We found that using simple peakfinders on high-frequency and noisy audio data results in poor results. A simple peakfinder will identify many more peaks than necessary. A simple minimum threshold or prominence approach does not work well either, as the found peaks become too sensitive to the actual magnitude of the signal, and thus louder keystrokes register as multiple peaks, whereas weaker signal register as no peaks. The common fix to this is to add a minimum distance between peaks. However, we found that trying to specify this parameter was very tricky, due to varying inter-character peak times (see Figure 3).

Instead, we adapted a heuristic algorithm that takes as input candidate peaks, and picks the "maximally distant from each other" peaks. Specifically, the algorithm considers the minimum amount of time it would have to move in either direction to find a value higher than it. The candidate peaks are sorted based on this metric, and the ones with the highest values are selected as peaks.

3.2 Windowing and Feature Extraction

Once we identify peaks, we propose windowing from the peak of the previous character to the next character. Although this may seem like an unorthodox choice that may introduce confounding peaks into the window, we note that only the actual character's peak will be conserved throughout all the windows. Furthermore, taking windows with (potentially) multiple characters has been done - for example, Giallanza et al. in [6] use 100ms windows and label them with just the first character in the window. We believe that windowing from the previous character to the next is a more foolproof way to make sure the whole peak is captured and is more adaptive in extreme cases (of very short inter-peak intervals and very long peaks). Features (in our case, cepstral coefficients) can be extracted from this window and will scale linearly with the size of the window.

A downside of our variable window sizes is that standard classifiers cannot be used, as they typically assume constant input size. We instead opt for using CNNs with global average pooling layers preceding readout layers. Convolutions are especially suitable in our case since the character may be anywhere in the window, so translation invariance is necessary.

3.3 Classifier

In an attack, an adversary would use a pre-trained model to classify keystrokes. As a result, we worked on training a classifier that would be able to accurately classify keystrokes on held out data. We investigated many CNN architectures for classification, all loosely based off of the ResNet architecture, and similar to the architecture in [6]. One of the architectures we investigated is shown in Figure 1, and is composed of two residual blocks, followed by a series of pooling and dense linear layers to map our convolution features to our character space.

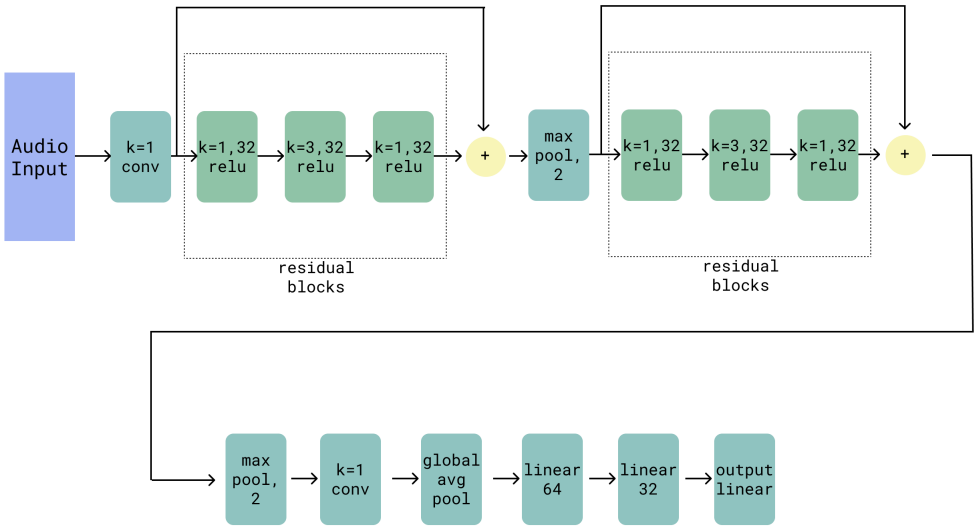

Figure 1: Our model architecture

One important detail is that because our input may be of variable size, which CNNs are not usually designed to handle, we use a global average pooling layer after the last convolution, which is applied per channel. The global average pooling serves to shrink our input size per channel into a single value. This ensures we are able to handle the variable size inputs (which window from the previous character to the next character).

4 Dataset

Of the three steps of our approach, only the last step involves a trained component, so we collected a dataset with the focus of training a classifier to identify keystrokes. To simulate the actual inputs this classifier would receive, we only gave it inputs that were windowed from a continuous stream of typing using the procedure described in Section 3.2. However, instead of using our custom peakfinder to isolate keystroke times from the audio stream, we used an online keylogger that gives timestamps for key-lifts so that we would have a ground truth and know that our data is not corrupted due to some inaccuracy on the peakfinder's part (since no peakfinder is perfect).

4.1 Collection

Multiple group members typed out long passages ranging from academic research papers to New York Times articles while recording themselves using Zoom in a relatively quiet environment. We chose to use actual articles (as opposed to hand-crafted text) to ensure naturalistic inter-character interval, as would be observed if one was trying to carry out this attack. This method, however, had the negative side effect of severe class imbalance. As a result, we filtered out all classes with less than 100 character samples. For the larger dataset, this yielded 20 classes, including letters, the space bar, and special characters like periods and commas.

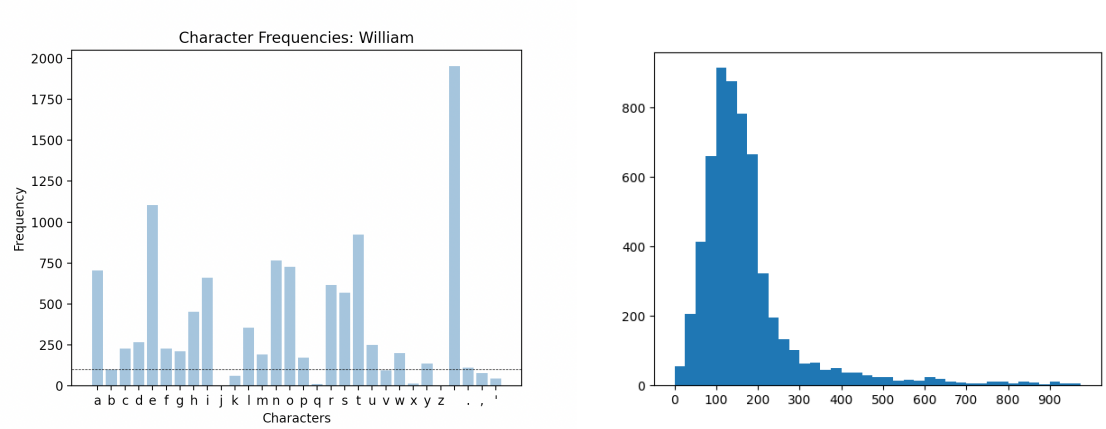

Figure 2: (From left to right) A graph detailing the frequency of each character, with a dotted lined to mark a frequency count of 100 and note which characters were included in our final dataset. A histogram noting the time lengths between keypresses in milliseconds and their frequency.

4.2 Alignment

After getting the recordings and keylogger files, we had to align them. Each typer was instructed to start off their typing with three presses of the letter 'a', separated by approximately 3 seconds. These 3 keypresses yielded very distinguishable peaks, which made for easy alignment of audio to keylogger files. After aligning, we noted that our ground truth labels were still slightly off potentially due to delays between keypresses and the actual audio being processed and recorded by zoom. However, we hoped that translation invariance in our classifier would be enough to combat this.

For our data collection, we used a keylogger web application that recorded when each key stroke was made (based on the release time of the key), and used Zoom's record audio feature. We then aligned the audio and keystroke logging data, which gave us our baseline ground truth.

5 Results

In this section, we present our findings, both positive and negative, on the effectiveness of such an attack. First, we focus on our proposed peakfinding method and discuss its benefits. Next, we dive deeper into the various different models and training hyperparameters we tried, and how these affected out-of-sample classification performance, which was measured as accuracy on a balanced test set containing 40 samples from each class.

5.1 Peakfinder

As our ground truth labels from the keylogger was misaligned, we found that there was no clear way of evaluating the peakfinder other than visually. Figure 3 presents the identified peaks using our method. Although not all peaks are perfect, the identified peaks largely capture the spikes corresponding to keypresses, potentially missing a few and identifying some that aren't actually peaks.

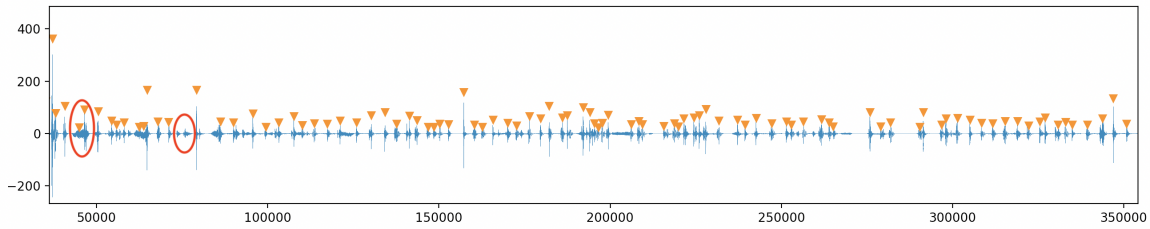

Figure 3: Peakfinder results. Most keystroke peaks are detected, mistakes shown in red circles red. Note that our approach does not completely mitigate false peaks and does not necessarily capture all peaks, but it has fewer mistakes than more naive alternatives. The model is robust (mostly) to varying peak heights and distances between peaks.

5.2 Keystroke Classification

As mentioned in Section 3.3, we used CNN-based architectures. We conducted many experiments, training over 50 different models (with varying architectures or learning parameters). All models were trained using the Adam optimizer with momentum 0.9. Class imbalance was accounted for in training using weighted resampling, and in testing by using a balanced test set. We varied batch sizes from 1 to 64, and found that batch sizes of 16 yielded the most stable results. The training losses and test accuracies for some runs are plotted in Figure 4.

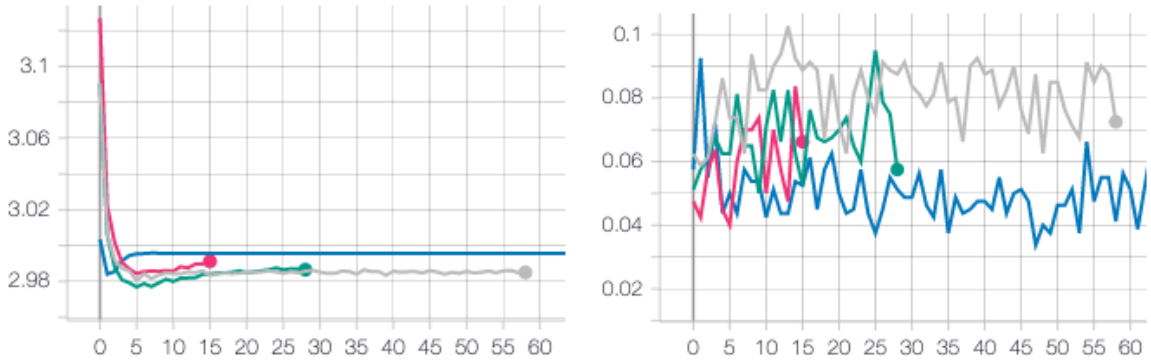

Figure 4: Training loss (left) and test accuracy (right) of a subset of our experiments. Earlier runs with other learning rates, batch sizes and architectures not shown. All shown architectures involve a single hidden readout layer of size 32 and increased convolutional filters to 64 instead of the 32 filters shown in 1. Blue indicates no dropout with a single, linear hidden layer of size 32. All runs not shown performed similarly to blue or worse. Pink indicates a run where dropout of 0.2 was added to the final hidden layer. Green indicates a run where the final hidden layer size was 64, dropout was 0.5, and a learning rate decay of 0.9 was added. Gray (best) indicates a learning rate decay of 0.8 and use of hidden layer size 32 with dropout 0.2. As seen, dropout and learning rate decay are critical to boosting performance of a classifier on this dataset.

In terms of architectures, we tried both adding and removing layers. We found that adding layers led the network to drop to chance level out-of-sample performance, potentially due to difficulties in training. In fact, we found that the best results occurred when we only used one hidden readout layer after the global average pooling operation. We also found that increasing the size of this layer (e.g. from 32 to 64) does not yield significant increases in performance, but the use of dropout does improve performance. We theorize that regularization such as dropout is necessary in our case of noisy input data.

Beyond architectures, the main parameters we varied were learning rate. Specifically, we examined various learning rates (ranging from 10^{-3} to 10^{-5}) and found that a learning rate of 10^{-4} was best. However, this learning rate still plateaus extremely quickly. To fix this early plateauing, which we later attributed to too large of a learning rate, we introduced an exponential decrease to the learning rate. We found that this led to stabler training and better out-of-sample performance.

Despite all these hyperparameter and architecture searches, the classifier's accuracy maxed out at 10% out-of-sample, which is significantly above the chance level (5%, or 1 out of 20), but not as high as we'd hoped. Especially compared to the top-1 accuracy of 41% in [6], we found our number striking. However, closer inspection of [6] indicates that using multiple modalities and using an error correcting language model both lead to significant enhancements and are likely the main reasons for their high reported accuracy. However, multiple modalities are not readily available in our case, and using a language model would not be effective in our case (which focuses on guessing passwords).

6 Challenges and Limitations

6.1 Challenges

We faced two main challenges throughout the project: establishing a perfect ground truth for our data, and constructing an optimal architecture (and picking optimal training hyperparameters) for our machine learning models.

Within the challenge of establishing a perfect ground truth for our data, the greatest hurdle came in verifying our audio to character matches and properly discarding outliers. Since we had over 15000 samples to work with, we could not go through and verify the proper matching of every keyboard click to keyboard character. Even if the first few characters of the sequence are aligned properly, that does *not* guarantee for us that all of the characters are properly aligned, since there could be a delay between either when a keystroke was logged, or when Zoom rendered it in the audio. Therefore, the best we could do for each audio file was to align the first few characters (see Section 4.2) and assume that the noise margin around each character would not be significant enough to disrupt our model.

We also faced a lot of challenges on the side of designing our architecture. We initially implemented the architecture mentioned in [4] without the motion data channel, but we found very poor results – nothing better than random chance. We then tried investigating alternative modifications on the architecture, and found some significant improvements (using a single hidden layer after the convolutions, using dropout, using learning rate decay). However, these improvements still only boosted our accuracy to 10%. With more computational power, more extensive grid searches and neural architecture searches could be performed. However, we stress that keystroke identification from the dampened audio in Zoom is not a trivial problem for adversaries to solve, involving simply copying a model off of an academic paper.

6.2 Limitations

In Section 4, we noted that our training and testing datasets only included characters with frequency counts greater than 100. As such, our results are only generalizable to the characters that were included in our datasets. In the interest of simplicity and attempting towards higher accuracy, we were unable to examine the keystroke tendencies of numbers, most special characters, and the Shift key. In particular, we were unable to accommodate the Shift key because of complications it caused in audio recordings. For most keys, the sound of pressing the key and releasing the key is concatenated into one noise that is identified as one peak in audio waveforms. However, for the Shift key, the purpose of the key divides the key press and release into two distinct peaks. While recording the timestamps of both the key press and release is not difficult, the release is oftentimes combined with the release of another key that is used in tandem. Two key releases are thus recorded at the same time, and the noise of these two actions are blended into one sound that make one key virtually indistinguishable from another. When generating our datasets, we ignored

both the Shift key and the keys that were immediately before and after it.

Our models may have limited generalizability to other keyboards, as all of our data samples were collected on MacBook laptop devices, meaning the microphone setup is same across all data samples. Our use of the built-in laptop keyboards for our recordings, with no variety in alternative desktops or keyboard extensions, could also lead to problems generalizing, as distance between our keyboards and laptop microphones is relatively the same. Having included samples of recordings with alternative keyboards would have given insight to identifying keystrokes with different baseline pitches and/or distances from microphones - we were unable to do so due to limitations in resources.

Finally, our data collection process only included gathering data samples from quiet environments with no background noise. Although our dataset does feature more noise (by virtue of dampened audio) than previous academic papers, it is still far from the typical scenario. For example, our model may not generalize to situations where someone is speaking and there is typing in the background, a common scenario in Zoom calls. We expect improvements to both the keystroke detection process and keystroke classification process would have to occur before this is possible.

7 Conclusion

7.1 Extensions

For future work, we would like to investigate the use of RNNs and Transformers for this task, since they tend to more naturally lend themselves to time-series data, as we have here. We also believe it would be fruitful to explore the combination of CNNs and Transformers, similar to how previous work has investigated the effects of stacking an RNN on top of a CNN [4]. To keep our models relevant for password detection, we would want to explicitly include typed nonsense words and phrases in our dataset to prevent the error correction module from just relying on language patterns (and instead use patterns in the time between keystrokes).

Additionally, since our work was done in a quiet environment across relatively similar keyboards, we would like to explore the generalizability to other noisier environments with different keyboards. Addressing the other limitations we listed above, expanding our project to accommodate more keyboard keys and a wider berth of laptop devices are avenues we could explore to improve our reach.

7.2 Final Remarks

Overall, we found the identification of keystrokes with audio recording-based keylogging a difficult feat. This is largely due to difficulties in generating accurate and appropriate datasets, as well as difficulties in finding appropriate parameters for our CNN models. Despite our disappointment in the low accuracies of our final results, the implication that Zoom recordings are relatively safe from audio-based keylogging attacks is comforting.

References

[1] K. Ali, A. Liu, W. Wang, and M. Shahzad, "Keystroke recognition using wifi signals," *ACM MobiCom*, pp. 90–102, 09 2015.

[2] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and ssh timing attacks." *USENIX Security Symposium*, vol. 10, 08 2001.

[3] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," *2004 IEEE Symposium on Security and Privacy*, 2004.

[4] L. Zhuang, F. Zhou, and D. Tygar, "Keyboard acoustic emanations revisited," *ACM TISSEC 1*, p. 3, 2009.

[5] A. Compagno, M. Conti, D. Lain, and G. Tsudik, "Don't skype type! acoustic eavesdropping in voice-over-ip," *ACM ASIACCS*, 2017.

[6] T. Giallanza, T. Siems, E. Smith, E. Gabrielsen, I. Johnson, M. A. Thornton, and E. C. Larson, "Keyboard snooping from mobile phone arrays with mixed convolutional and recurrent neural networks," *ACM Interact. Mob. Wearable Ubiquitous Technology*, vol. 3, no. 2, p. 22, 06 2019.