

# Information Leakage through Packet Lengths in RTC Traffic

Jaiden Fairoze<sup>1</sup> and Peitong Duan<sup>1</sup>

University of California, Berkeley  
{fairoze, peitongd}@berkeley.edu

**Abstract.** Modern real-time communication (RTC) depends on efficient audio and video compression to minimize bandwidth requirements. While the codecs are certainly up to the task, the combined effect of live data, lossy compression, and length-preserving encryption leads to the possibility of leakage: the lengths of encrypted packet sequences can leak information about the underlying, unencrypted data. In this work, we measure leakage in real-world RTC platforms and analyze whether information is recoverable. We survey a range of real-world RTC platforms (along with various configurations of each platform) for their propensity to leak data through encrypted packet lengths. We then conduct an in-depth study on Zoom and collect 268,392 network traces amounting to over 786.8 hours of recorded media. We use this data to train models to automatically recover information from packet lengths. This is achieved by leveraging existing action recognition datasets, along with a new dataset of virtual meeting recordings. We train neural networks for action recognition over encrypted packet lengths rather than the source data itself. We find that under certain conditions, our models significantly outperform random selection, in effect quantifying leakage for the given learning problem.

**Keywords:** Information leakage · Real-time communication · Measurement

## 1 Introduction

Internet-based communication is more prevalent than ever. As a result of the COVID-19 pandemic, audio and video conferencing tools saw unprecedented growth [44]. Real-time communication (RTC) platforms have been widely adopted for corporate use, healthcare visits, recreational purposes, and beyond. In other words, the data handled by RTC platforms has ballooned in size and sensitivity, and platform security is paramount. For instance, conversation confidentiality is particularly critical given the prevalence of telehealth. If information about the plaintext can be extracted in any way, the consequences for compliance (e.g., HIPAA) would be severe.

*Current security practices.* The usual defense mechanisms include encrypting data at rest and in transit. Encryption of static data in transit is usually straightforward because the plaintext is entirely *known*. Encryption is more challenging when the plaintext is dynamic or freshly streamed, as is the case in RTC. This problem is compounded by codecs that exaggerate information leakage through packet lengths. In fact, such attacks are known to be theoretically feasible in specific codec and encryption configurations in both the audio [55,54,2,51] and video [36,18,3] domains.

These attacks are possible because encrypting data as it comes produces burst patterns [36] that reveal bits of information about the plaintext. Even more concerning, our formal notions of security break down in the streaming setting. Standard security definitions assume the plaintext and ciphertext are the same length: this is clearly violated if the plaintext is not entirely known at the time of encryption.

In standard secure encryption, the adversary chooses two plaintext messages  $m_1$  and  $m_2$  such that  $|m_1| = |m_2|$  and receives the encryption of one of them. The adversary’s goal is to determine which message was encrypted. Alternative security notions exist such as “real-or-random”-style definitions where the adversary instead distinguishes between an encryption of a chosen  $m$  and a random string of the same length. Another variant is simulation-based security, where security is implied by the existence of an efficient simulator that produces the same distribution of ciphertexts as the real encryption scheme. Critically, all of these definitions require that the ciphertext length is the same as the plaintext length, which does not hold in the streaming setting.

In the RTC context, the direct composition of variable bit rate (VBR) encoded data and length-preserving encryption (LPE) (say, the Advanced Encryption Standard [6]) is known to be particularly devastating. Prior work has demonstrated that under simulated conditions, the language spoken in a conversation [55], the identity of speakers [2], and even transcript extraction [54,51] is possible from only network-level data (i.e., encrypted packets) under VBR encoding and LPE.

*Our work.* In this project, we initiate the study of real-world RTC leakage. We ask the following question:

*Can we extract information about the underlying plaintext from ciphertext packets produced by live RTC applications?*

Our key approach toward answering this question is to emulate pairwise RTC communication exactly: we set up *real* RTC calls upon which we perform our experiments. This (re)produces the network packets that the RTC platform would generate if the communication occurred naturally. We do this by spoofing both the audio and video devices in software and feeding the pre-recorded media into the respective devices. From the RTC client’s perspective, it is as if the audio and video comes live from a recording device (i.e., microphone or camera). This means the packets we capture are genuine encrypted packets from a given RTC platform and the source data of our choosing. While our emulation is exact

for pairwise communication, we use the same setup to analyze data that involves more than two participants: we discuss the implications of this in addition to the setup’s limitations in [Section 3.2](#).

## 1.1 Our Contributions

We explore the possibility of leakage through packet lengths—for the question above, we find that the answer is “yes”: we can extract information about plaintext data from sequences of ciphertext packets. Our contributions include:

1. **Semi-automated packet collection pipeline.** We first develop a semi-automated data collection framework to repeat our experiments across different datasets and platforms. We gathered 268,392 network traces (capture files) across 7 datasets and 9 platforms, amounting to over 786.8 total hours (334.6 GiB) of raw packet data.
2. **Characteristic analysis.** Using the data gathered from our pipeline, we provide a heuristic security analysis for nine major RTC platforms (listed in [Table 1](#)). We stress test each platform (under specific settings) with high- and low-entropy synthetic data to evaluate the audio and video codecs’ responses. The test results serve as an indicator of each platform’s vulnerability to audio and video leakage. Ultimately, we select Zoom for targeted analysis due to the video codec’s response to our tests (see [Figure 2](#)), coupled with the platform’s widespread use. Refer to [Section 4](#) for more detail about the characteristic analysis experiment.
3. **Information leakage analysis.** We focus on Zoom for in-depth leakage analysis along two lines of investigation:
  - (a) *How well can we classify human actions from existing ML datasets streamed over Zoom?* The relevant datasets are HMDB51 [\[25\]](#), UCF101 [\[40\]](#), Charades [\[37\]](#), and Something-Something [\[15\]](#). For each dataset, we find that standard neural networks (LSTM, GRU) trained on the dataset learn non-trivial information from packet sequences. The largest advantage over random (i.e., the most leakage) we measure is 27.3% test accuracy over 101 classes (vs. baseline of 0.99%) (see [Table 2](#)). Note that with strong security, one should not be able to use packet lengths to gain any classification advantage.
  - (b) *How well can we classify natural video conferencing data on Zoom?* We scraped public recordings of work calls from YouTube, re-streamed them over Zoom, and collected the encrypted packets. We use this custom dataset to again train standard neural networks (LSTM, GRU) to predict classes on unseen data. In this case, the classifier is trained to assign a class out of: screen sharing, single participant speaking, and a gallery view of all participants. Again, we identified leakage was present: we were able to obtain test accuracy of 65.23% (vs. baseline of 33%).

## 2 Background

*Using Packet Lengths for Fingerprinting* Prior work has shown fingerprinting is possible for video-based encrypted streams. We refer the reader to [Section 6.1](#), a survey of major fingerprinting applications. These attacks are made possible by a varying bitrate. Previously, video fingerprinting was conducted on platforms that use Dynamic Adaptive Streaming over HTTP (DASH) [36,18,27] or HTTP adaptive streaming (HAS) [3]—these protocols are optimized for one-way data streaming. This is often appropriate for applications with a public or readily-accessible stream source such as YouTube, Netflix, or Twitch. A malicious actor observing the network could launch the following attack: Say the actor wishes to know whether her target is watching Video A, Video B, or Video C. She can observe the sequence of encrypted packets delivered to her own machine when viewing or streaming each source herself. Then, to spy on her target, she can compare the known packet sequences to the target’s current network activity and deduce whether it matches A, B, C, or none. This fingerprinting attack functions for two key settings that we define below: *public-static* and *public-dynamic*.

*Public-static* In this setting, e.g. a YouTube video [36], the source data is both public and static. Public data implies the attacker has the ability to produce encrypted packet sequences for the source data, i.e., she has access to the raw fingerprint of each video of interest. Static data implies the fingerprint is stable: it does not change over time. Once a fingerprint is obtained, the attacker can simply look for it “in the wild.”

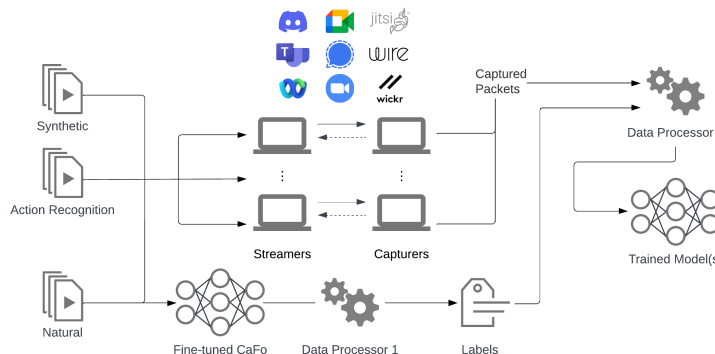
*Public-dynamic* In this setting, e.g. a Twitch stream [19], the source data is still public, but it is now dynamic. The attacker can still access raw fingerprints by viewing the livestream herself, thereby observing the live encrypted packet sequence (fingerprint). The dynamic setting is more difficult because the attacker must now monitor her targets and the source data simultaneously.

*Private-dynamic* In both scenarios above, it is relatively straightforward to launch a fingerprinting attack given the publicly accessible source data. In this paper, we tackle a harder variant of the problem under the private-dynamic setting: we attempt to learn information from encrypted packets without knowing what source produced it. Intuitively, private data changes the problem from matching exact fingerprints to clustering similar fingerprints: since the attacker is no longer able to view the target data herself, she must now learn from similar media to make predictions about encrypted data she has not seen before.

We set out to tackle this classification task in the private-dynamic setting over RTC platforms.

## 3 Experimental Pipeline

*Terminology* Throughout the paper, we use the following terms. *Dataset* refers to a collection of *source* files: for example, the UCF101 dataset contains a source



**Fig. 1.** High-level overview of our end-to-end evaluation pipeline. Synthetic data depicts our high- and low-entropy multimedia. Action recognition depicts the ML datasets that we transformed for our leakage study. Natural depicts our custom Zoom conferencing dataset and requires an extra inference step to generate labels.

`v_Kayaking_g22_c03.mp4`. Each source within each dataset is streamed over an *app*, where an *app* is a configuration of an RTC platform specified in [Table 1](#). For example, `zoom-auto` for Zoom with automatic noise cancelling enabled.

*Overview* We designed and implemented a semi-automated end-to-end pipeline to collect and analyze encrypted packets from an array of RTC platforms for our experiment. The pipeline is extendable to any RTC platform that runs a desktop operating system. We describe the primary components of the pipeline at a high level and refer the reader to [Figure 1](#) for a visual overview.

- (1) **Dataset preparation.** The first step is preprocessing input datasets. This ensures that (a) metadata is consistent across different datasets and (b) multimedia is encoded efficiently for the experimental environment. This step is omitted from [Figure 1](#) for brevity.
- (1.1) **Automatic labeling.** Our natural dataset consisting of recordings of real meetings requires an additional processing step: a purpose-built neural network is used to automatically break up and label long network captures into manageable labeled chunks. This is represented by “Fine-tuned CaFo” and “Data Processor 1” in [Figure 1](#).
- (2) **Encrypted packet streaming and capturing.** For a given RTC platform, automated scripts stream and capture encrypted packets for each source in each dataset. This is performed between two distinct physical devices over a wireless network. This step is depicted by the “Streamers” and “Capturers” in the middle section of [Figure 1](#).
- (3) **Encrypted packet processing.** Once all encrypted packets are captured, various filters are applied to de-noise and normalize network captures such that packets unrelated to the RTC platform are removed, and RTC-related packets are signal-processed to emphasize bitrate changes (i.e., emphasize

burst patterns). At this point, the data is ready for analysis. This step is represented by “Data Processor 2” in Figure 1.

- (4) **Model training.** We train DL models to perform specific classification tasks depending on the input dataset. We use the training results to compare leakage across platforms for each dataset. The model’s ability to learn (measurable via validation loss) is correlated with leakage. This step is represented by “Trained Model(s)” in Figure 1.

### 3.1 RTC Platform Selection

	Capturer Version	Streamer Version	Protocol	Audio Codec	Video Codec
<i>Discord</i>	Stable 147045 (6ba38a9)	Stable 147045 (6ba38a9)	DTLS-SRTP[46]	Opus	H.264
<i>Google Meet</i>	Unknown	Unknown	DTLS-SRTP[14]	Opus	VP9, VP8 (screen sharing)
<i>Jitsi Meet</i>	2.0.7648	2.0.7648	Custom SFrame[5]	Opus	VP8 (default), VP9, H.264
<i>Microsoft Teams</i>	1.0.0.22073101005	1.0.0.22073101005	DTLS-SRTP[28]	Satin (Opus)	H.264
<i>Signal</i>	5.58.0 production	5.58.0 production	Custom SFrame[42,11]	Opus	VP8
<i>Cisco WebEx</i>	42.9.0.23494	42.9.0.23494	Custom SFrame[4]	Opus	H.264, AV1
<i>Wickr</i>	Windows v5.106.15 build 1	Linux v5.106.14 build 1	—	Opus	—
<i>Wire</i>	Version 2022.06.30.13.51	Version 2022.06.30.13.51	DTLS-SRTP[52]	Opus	—
<i>Zoom</i>	Version 5.11.10 (4400)	Version 5.11.1 (6602)	DTLS-SRTP[58]	Opus	H.264

**Table 1.** Summary of RTC platforms evaluated. Version numbers represent the earliest version used—applications were kept up-to-date through our data gathering period. The protocol for a given platform comes from public documentation. If none was available, we leave the cell blank. The audio and video codec columns are not exhaustive: only primary and default codecs listed. Namely, some platforms support other legacy codecs such as G.711 and G.722. We exclude these for brevity. Smaller platforms have limited documentation regarding codec support. Note that Satin is a superset of Opus.

We cast a large net over a wide range of RTC platforms as presented in Table 1. These platforms were selected for their popularity (e.g. Zoom) and/or their focus on security (e.g. Signal). Recall we define different configurations of the same RTC platform as an *app*, and consider them independently in the experiment. We do not tweak or constrain the operation of any RTC platform beyond what can be achieved through the honest “Settings” interface. In particular, we assess each platform for settings with direct impact on audio or video bandwidth, such as noise cancellation or VBR/CBR toggles. Overall, we streamed over 786.8 hours of multimedia during the experiment, resulting in 334.7 GiB worth of network packets.

### 3.2 Automated Packet Collection

We define two entities: the *Streamer* machine and *Capturer* machine. The Streamer is responsible for streaming encrypted multimedia data to the Capturer, whose job is to capture the relevant packets. For a given app, Streamer and Capturer join a “call.” The Streamer’s job is to spoof the video and audio devices (i.e., the native microphone and webcam feed) with custom multimedia. The Capturer’s job is to capture the packets generated by the spoofed data.

*Streamer* We developed a custom OBS Studio [39] script for the Streamer device. OBS, armed with the script, allows us to feed the desired video media through a virtual webcam and control it systematically. For audio, we use the PulseAudio sound server on Linux to reroute system monitoring audio into a virtual microphone, meaning audio playback (such as the OBS playback) is rerouted into the virtual microphone. The script is written in Python and coordinates with a Capturer-side program using `asyncio` and `websockets`. The Streamer signals to the Capturer when to capture packets (i.e., once streaming starts) and sends relevant metadata. For each app, we establish a connection between the Streamer and Capturer to perform automated packet collection for every dataset.

*Capturer* In tandem with OBS on the Streamer side, a corresponding headless program runs on the Capturer side to automate packet collection. It is a Python program that listens for signals from the Streamer to start or stop capturing packets. It additionally receives metadata to allow the Capturer script to appropriately label captures. The program captures network packets using the `tshark` command line tool from the Wireshark [53] suite.

*Experimental procedure* Before we started the experiments, we installed a fresh copy of Ubuntu 22.04 on both the Streamer and Capturer machines along with our tooling. We then do the following for each app and dataset combination:

1. Upload the relevant dataset sources to the Streamer device.
2. Pair the two devices over the RTC platform of choice, i.e., place them in a call. If the dataset has one stream (only audio or only video), mute the unused channel.
3. Start the capturer script on the Capturer device.
4. Run the OBS script on the Streamer device. Ensure that OBS’s virtual camera is the default webcam and that system sound is in monitor mode, guaranteeing the RTC platform will use the virtual inputs. This begins the capture process. It will run automatically until all sources in the dataset have a corresponding network capture.

*Limitations* The primary goal of our emulation setup is to reproduce genuine packets for selected source data. While the emulation is accurate for all action recognition datasets, there are limitations with our Zoom conferencing dataset.

First, our data collection pipeline only supports unidirectional data streaming, i.e., streaming from a single source device to a single destination device. In a real Zoom call between two participants, data is bidirectional and each individual stream is selectively forwarded. We claim the unidirectional setup is sufficient due to the following approximation: for two media streams  $A$  and  $B$  where  $\text{encode}(x)$  returns the size of  $x$ ’s encoding in bits and  $\oplus$  denotes side-by-side (i.e., stacked) concatenation, if  $|(\text{encode}(A) + \text{encode}(B)) - \text{encode}(A \oplus B)| < \delta$  for a sufficiently small  $\delta$  (meaning the encoder doesn’t have “too many” opportunities to compress both streams together), then  $\text{encode}(A) + \text{encode}(B) \approx \text{encode}(A \oplus B)$ .

To reproduce a bidirectional and selectively forwarded RTC stream in a unidirectional manner, we mix the streams by concatenating them “side-by-side”.

By the approximation above, we are still able to measure burst patterns so long as the burst signal of any one stream is not completely eliminated by reencoding: the bidirectional case corresponds to the LHS, and our unidirectional (mixed) setup corresponds to the RHS.

Second, many of the Zoom recordings that we use are from calls that involve more than two participants. Since our setup only involves two devices, it is an imperfect reproduction. However, we claim this discrepancy is minimal due to the same reasoning as above—only this time, we mix more than two streams.

In summary, our collection architecture is faithful to packets produced by server mixing: as long as the mixing approach does not eliminate the burst signal of any one stream, the burst contribution of any individual stream is measurable through packet lengths. However, other architectures exist, such as selective forwarding. We leave it to future work to design and implement a collection framework that is closer to these architectures. This, we expect, should allow for more accurate measurement of leakage through packet lengths.

## 4 Phase 1: Platform Leakage Characteristic Analysis

The goal of Phase 1 is to identify real-world RTC platforms that are likely to leak significant information through packet lengths. Suppose we have two videos: Video A and Video B. The key idea is: if no information is leaked from encrypted packet lengths, then the total packet size from streaming A then B should be equal to that of streaming B then A. We formalize this idea and systematically evaluate a wide range of RTC platforms.

*Media generation* Many prior works studied length leakage in an idealized setting [55,54,2,51]—they analyzed the output of (mostly audio) codecs under *chosen* conditions. On real RTC platforms, we do not have fine-grained control over codec choice, codec parameters, or how codec payloads are encrypted. In Phase 1, we analyze the packets produced by live RTC applications by leveraging synthetic data (as opposed to genuine media input such as webcam footage or microphone capture) designed to test various aspects of the underlying codecs. For both audio and video media, we generate extreme synthetic data that makes use of (a) minimal entropy data (a white background for video and silence for audio) and (b) maximal entropy data (uniformly random video and audio). We use this data to empirically measure the codec’s response to rapidly changing bitrates. This is done by concatenating high entropy data and low entropy data (and vice versa). At the point of concatenation, the media codec is “shocked” by a rapid change in bitrate and its response can be measured. The codec goes through an adjustment period in which it optimizes itself for the new bitrate data—this optimization process can leak information. That is, the fact that bitrate has rapidly changed says something about the underlying plaintext.

Let  $H_t^{\text{media}}$  denote the  $H \in \{U, W\}$ -entropy media file of duration  $t$  seconds and type  $\text{media} \in \{\text{audio}, \text{video}\}$ . Note that  $U$  denotes maximum entropy (uniformly random) media and  $W$  denotes minimum entropy (white or silent) media.

Let  $H_t^{\text{media}} \parallel H_{t'}^{\text{media}'}$  denote the head-to-tail concatenation of two media files. The  $\parallel$  operator preserves order. Then, for  $\text{media} \in \{\text{audio}, \text{video}\}$ , we specify our synthetic data accordingly:

$$\begin{aligned} U &:= U_{120}^{\text{media}} & UWUW &:= U_{30}^{\text{media}} \parallel W_{30}^{\text{media}} \parallel U_{30}^{\text{media}} \parallel W_{30}^{\text{media}} \\ W &:= W_{120}^{\text{media}} & WUWU &:= W_{30}^{\text{media}} \parallel U_{30}^{\text{media}} \parallel W_{30}^{\text{media}} \parallel U_{30}^{\text{media}} \\ UW &:= U_{60}^{\text{media}} \parallel W_{60}^{\text{media}} & WU &:= W_{60}^{\text{media}} \parallel U_{60}^{\text{media}} \end{aligned}$$

Later in the paper, we drop  $\text{media}$  when it is inferable from context.

All synthetic audio sources were encoded with WAV [22] – a lossless audio codec. All synthetic video sources were encoded with VP9 [34] in lossless mode with FFmpeg using the following configuration: `ffmpeg -i {input} -c:v libvpx-vp9 -lossless 1 -async 1 {output}`

*Data collection* We set up pairs of RTC devices to programatically stream the synthetic datasets over the nine RTC platforms mentioned in Table 1 following the procedure described in Section 3.2.

For each platform, we tested the following apps (different configurations of the same RTC platform):

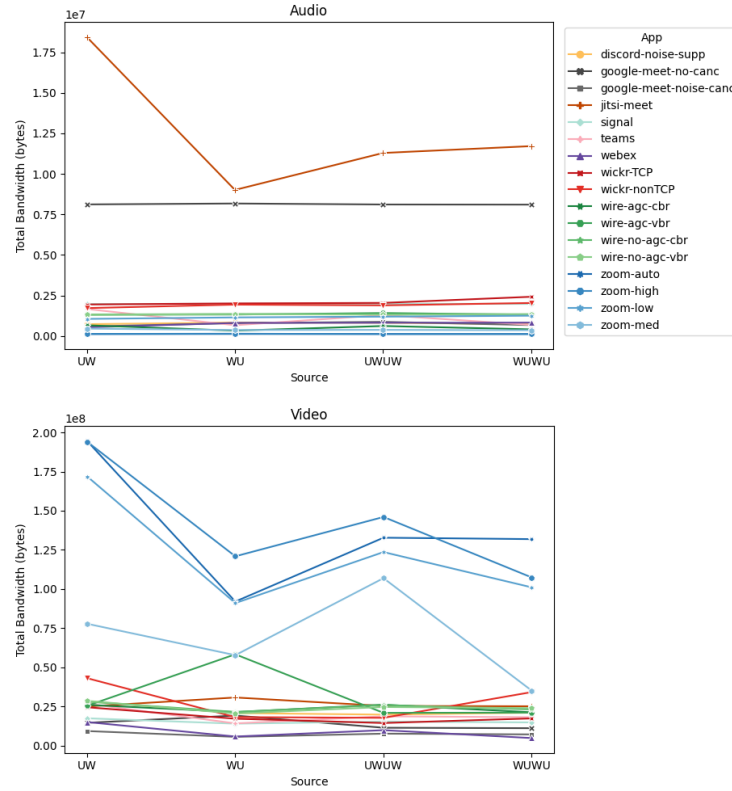
1. `discord-noise-supp`: Discord with Krisp noise suppression enabled.
2. `google-meet-(noise-canc|no-canc)`: Google Meet with noise cancellation enabled and disabled, respectively.
3. `jitsi-meet`: Jitsi Meet with performance setting at highest quality.
4. `signal`: Signal with default configuration.
5. `teams`: Microsoft Teams with default configuration.
6. `webex`: Cisco WebEx with default configuration.
7. `wickr-(TCP|nonTCP)`: Wickr with TCP mode enabled and disabled, respectively. Note that TCP disabled means that UDP mode is used.
8. `wire-(agc|no-agc)-(cbr|vbr)`: Wire with automatic gain control turned on (off), CBR enabled (disabled), and VBR disabled (enabled), respectively.
9. `zoom-(low|med|high|auto)`: Background noise filtering set to low, med, high, and auto.

For all platforms, we enable HD video and preserve aspect ratios when possible.

*Results* With the captured packets for each app, we are able to see the total length of data transmitted over the network and measure the magnitude of information loss from the codec’s compression. This is shown in Figure 2.

Recall that we expect a horizontal line for an app if it does not leak any information through its encrypted packets. Examining Figure 2 more closely, we can see two categories of lines:

1. Relatively stable lines for the majority of apps, which indicates low leakage. Additionally, these apps have total bandwidth close to their mean bandwidth over the four UW, WU, UWUW, and WUWU sources. At best, `signal` has all four bandwidth values within 0.51% of its mean in the audio setting, and `wire-agc-cbr` has all four within 0.10% in the video setting.



**Fig. 2.** Total bandwidth measurements for various permutations of high-and low-entropy audio (left) and video (right) over a range of RTC apps. The expectation for an app with zero leakage is a perfect horizontal line. Refer to the “Data collection” paragraph in [Section 4](#) for app descriptions.

2. Jagged lines for a handful of apps. These indicate higher leakage and suggest the apps are compressing data in an asymmetrical, lossy fashion.
  - (a) Among the jagged lines, it is common that sources prefixed with U result in larger-than-expected total packet lengths, indicating that it takes considerable time for the codec to adjust between high and low bitrate data. In the audio setting, *jitsi-meet* and *teams* exhibit this pattern, deviating by 46% and 55% from their respective means. In the video setting, all *zoom* variations exhibit this pattern: even the flattest *zoom* line deviates from the mean by 37%, and the others even more so. In general, the audio graph is more stable than its video counterpart, which is expected due to the larger amount of data transferred over video. This further emphasizes the effect of codec sensitivity.
  - (b) In *jitsi-meet* audio and *zoom-auto* video, we see that UW and WU have significant bandwidth magnitude differences, whereas we see rela-

tively equal bandwidth totals for UWUW and WUWU. This suggests the codec may be able to better adapt when there are repeated bitrate shocks (three bitrate transitions rather than one).

- (c) There are a few outlying, less-pronounced jagged lines in the video graph which do not exhibit the pattern described above: `wickr-nonTCP` and `wire-agc-vbr`. The largest difference from the mean across all apps was 86% for `wire-agc-vbr`.

Next, we discuss the effect of different settings across the apps. We tested each setting with all other parameters held equal.

1. **Background noise reduction.** We see a large average bitrate reduction for Google Meet in the audio setting when noise cancellation is enabled: the total packet length mean across the four audio sources was 8.13 MB vs. 0.73 MB. Similarly, Zoom decreases in bandwidth as noise filtering strength increases. Notably, the bandwidths of `zoom-auto` and `high` were close (124.20 MB vs. 129.49 MB respectively), suggesting that, given the choice, Zoom automatically selects the high noise filtering preset for our synthetic data. `zoom-med` used 389.93 MB of bandwidth and `zoom-low` used 1,162.32 MB total bandwidth. The results are shown in the dominant `google-meet-no-canc` line over its `noise-canc` counterpart, as well as the increasing order of `zoom-high`, `med`, and `low`. These observations suggest the noise filters are effective—on both platforms, the respective filters are able to detect the presence of uniform noise and remove it from the signal.
2. **TCP vs. UDP.** Compared to Wickr with TCP enabled, we expect Wickr over UDP to reduce total bandwidth since the platform does not need to spend as much bandwidth for non-payload packets. We find the audio setting is consistent with this expectation: UDP uses less data (1,896.46 MB) than TCP (2,104.48 MB). However, in the video setting, we saw the opposite effect: UDP used 28,240.65 MB vs. 18,314.24 MB for TCP. We conjecture the UDP option may support a larger maximum bitrate, and the uniform video streaming would likely reach the limit.
3. **CBR vs. VBR.** Wire enables users to toggle between CBR and VBR—it was the only RTC platform that provided such a control. In the audio setting: (a) With AGC enabled, switching from VBR used 1,343.67 MB vs. CBR’s 505.45 MB. (b) With AGC disabled, bandwidth costs are relatively unchanged: 1,324.75 MB for VBR vs. CBR’s 1,342.19 MB. In the video setting: (a) With AGC enabled, switching from VBR to CBR significantly reduces bandwidth costs: from 31,388.27 MB to 23,676.94 MB, respectively. This was unexpected since AGC should, in theory, have no effect on the amount of transmitted data. We again conjecture that another variable may have been at play. (b) With AGC disabled, bandwidth costs are relatively unchanged with 24,371.11 MB for VBR and 24,594.48 MB for CBR.
4. **Automatic gain control vs. manual gain control.** We tested the effect of automatic gain control (AGC) for audio on Wire under both VBR and CBR conditions. Examining the data from [Item 3](#) along the AGC axis, we observe that in the VBR case, toggling AGC had little to no effect on bandwidth:

both VBR with and without AGC used on the order of 1,300 MB bandwidth. However, in the CBR case, turning AGC on reduced total bandwidth from 1,342.19 MB for CBR without AGC to 505.45 MB with AGC.

Our findings from Phase 1 show us the magnitude of leakage in the video setting is far larger than the that of the audio setting. Hence, we focus on video for our Phase 2 analysis since it is easier to measure burst patterns. Zoom exhibits the strongest indication of leakage in the video setting, so we proceed with targeted leakage analysis on this platform.

## 5 Phase 2: Learning from Lengths

We focus on Zoom for video-based analysis and begin Phase 2 to thoroughly assess the information leakage. We measure leakage by training neural networks to see whether and how much they can learn from encrypted packet lengths alone: if they perform better than random, leakage is present.

### 5.1 Experimental Setup

*Media generation and data collection* We use a range of well-studied action recognition datasets in addition to a custom dataset containing labeled Zoom conference recordings. They are:

1. HMDB51: Kuehne et al. [25] manually annotated around 7,000 video segments of movies and YouTube clips in 2011. They were classified into 51 categories that represent common verbs such as “jump,” “run,” and “sit” to name a few examples.
2. Something-Something V2 (S-S): Goyal et al. [15] curated a dataset designed to help neural networks learn about trivial physical phenomena in 2017. Their database contains over 100,000 video clips classified into 174 classes. Each class is a descriptive English caption. For example, “Pushing a green chili so that it falls off the table.”
3. UCF101: Soomro, Zamir, and Shah [40] compiled a dataset of 101 action classes. It contains over 13,000 distinct clips totaling to around 27 hours of video data. The data is user-uploaded footage to YouTube. It includes varying camera motions and background footage per class.
4. Charades: Sigurdsson et al. [37] annotated 9,848 common daily human actions. The authors claim the actions are not commonly found in public sources such as YouTube because they are mundane actions. The videos are, in general, longer than the other action recognition datasets; their average length is about 30 seconds.
5. Conferencing: In addition to using existing datasets, we also construct a new dataset consisting of recorded conference calls over Zoom. We scraped the data from YouTube and labeled it automatically using a fine-tuned Transformer model [57]. We defined three scenarios that are likely to cover all frames of an active Zoom call, i.e., the call is always in one of the three scenarios. In particular, we labeled instances of screen sharing, participants speaking, and gallery view of all participants.

*Classification tasks* For the existing action recognition datasets, we attempt the same action recognition task for which the dataset was designed, except we perform it over encrypted packets that form the dataset’s training instances. For our custom Conferencing dataset, the goal is to classify the meeting scenario for a given timeframe. We design three unified neural net architectures to perform classification over encrypted packets. We retrain each model for each specific dataset and classification problem.

## 5.2 Model Architecture and Training

We use sequence-to-one models to classify sequences of video packet lengths. Specifically, we use three different encoders, each with a recurrent architecture, to embed the packet sequences of varying lengths. Each encoder then generates a fixed-length embedding of the packet sequence for input to a classifier that predicts the label for the video sequence.

*Capture processing* We filter the captures to retain only the packets that travel from the Streamer to the Capturer over UDP. We throw away packets that do not originate from the known IP addresses of the Streamer or Capturer.

Our goal is to train on sequences of packet length data. Let  $S = \{(p_i, t_i) | i = 0, \dots, n\}$  denote the  $(n + 1)$ -long sequence of (packet length, timestamp) pairs corresponding to a single label, i.e., it is a single training instance. Rather than training on sequence  $S$  directly, we instead train on  $S' = \{(\delta_{p_i}, \delta_{t_i}) | i = 1, \dots, n\}$  where  $\delta_{p_i} = p_i - p_{i-1}$  and  $\delta_{t_i} = t_i - t_{i-1}$  respectively. We do this to assist the model in learning generic patterns among network traces and prevent it from memorizing specific sequences of packets for a given arrival sequence. We find this approach improves performance on unseen data and reduces overfitting.

*Preparing the Conferencing dataset for training* We use 20-shot fine-tuned CaFo [57], a state-of-the-art low-shot image classifier, to automatically label frames spaced one second apart in long Zoom recordings. The Zoom recordings (often greater than 30 minutes) are much longer than media from other datasets (less than 1 minute), so we algorithmically split up long packet captures into smaller chunks. The quality of the chunks relies on the quality of CaFo’s predictions. While fine-tuned CaFo had perfect test accuracy on 294 hand-labeled frames from our dataset, misclassifications are still possible. These errors would propagate to the final model—to minimize their effect, we look for long contiguous sections of the same CaFo-assigned class and drop sections that are below a threshold. This is a heuristic filter based on the relative stability of a Zoom conference call: each class (e.g., screen sharing) is likely to last for many frames (seconds to minutes). In practice, we drop all frames that are not in a contiguous section of at least 5 seconds—the probability that the model misclassifies all frames in the section is low. Another option for mitigation would be to increase the number of shots when fine-tuning CaFo, but we found this to be unnecessary.

*Models* We explored three different model architectures for the encoder:

1. **LSTM with Attention.** This model uses an encoder with two stacked LSTM layers. After reading in the entire sequence, all the hidden states of the second LSTM are passed to an attention layer to generate a weighted mean that better captures the entire sequence. This is used as the embedding of  $S'$  and is fed into the classifier.
2. **LSTM.** This model also uses two stacked LSTM layers for the encoder. We drop the attention layer described above.
3. **GRU.** This model replaces the two stacked LSTM layers with two stacked GRU layers.

GRUs have a simpler architecture, so they have faster training time and less memory usage. However, they generalize worse than LSTMs, and we wanted to see how their performance compares. Finally, attention has been shown to improve performance, especially in long sequences, as it enables the model to focus on relevant parts of the sequence when making predictions.

Each encoder reads in one element of  $S'$  at a time, and takes in its  $\delta_p$  and  $\delta_t$  values as features. Each classifier consists of a single feed-forward layer followed by the final output layer, corresponding to the set of classes. The size and number of layers for both models were tuned via a hyperparameter sweep. All models were implemented and trained with PyTorch [32].

For each model and dataset, we use 80 percent for training, 10 for validation, and 10 for test. We train the entire model end-to-end with the Adam optimizer, a learning rate of  $3e-4$ , and a batch size of 64. We save the model with the best validation loss and evaluate it on the test set.

### 5.3 Results

<i>Dataset</i>	Class Count	Baseline	<i>LSTM with Attention</i>		<i>LSTM</i>		<i>GRU</i>	
			Accuracy	Advantage	Accuracy	Advantage	Accuracy	Advantage
<i>HMDB51</i>	51	1.96%	9.02%	7.06%	7.10%	5.14%	8.45%	6.48%
<i>UCF101</i>	101	0.99%	27.30%	26.31%	6.17%	5.18%	22.39%	21.40%
<i>Charades</i>	157	0.64%	1.16%	0.52%	0.19%	-0.44%	0.58%	-0.06%
<i>Charades (maximized)</i>	91	1.10%	2.70%	1.60%	2.08%	0.98%	2.12%	1.03%
<i>S-S</i>	174	0.57%	0.90%	0.32%	1.45%	0.87%	0.85%	0.27%
<i>S-S (maximized)</i>	105	0.95%	4.47%	3.52%	3.81%	2.86%	4.61%	3.65%
<i>S-S (reabeled)</i>	48	2.08%	5.07%	2.99%	4.43%	2.35%	4.51%	2.43%
<i>S-S (reabeled, maximized)</i>	13	7.69%	18.49%	10.80%	16.89%	9.20%	16.71%	9.02%
<i>Conferencing</i>	3	33.33%	64.62%	31.29%	65.23%	31.90%	61.30%	27.97%

**Table 2.** Maximized datasets have been optimized for the number of training instances at the cost of distinct classes. S-S (reabeled) denotes an alternative labeling for the S-S dataset—it groups similar motion labels into larger sets. Baseline performance is the expected performance for a classifier that assigns classes randomly (i.e. the inverse of the number of classes). This is the best possible performance if there is no leakage. We define advantage as the difference between baseline performance and test accuracy.

Table 2 shows each model’s performance on each dataset. The baseline performance shown in the table is computed from  $\frac{1}{\text{class count}}$ . This baseline is the expected performance for a random assignment of classes, i.e., no leakage. We study

performance via an advantage measure: the difference between the model’s performance and random class assignment—the higher the advantage is, the better the model is relative to its baseline. Of our three models, LSTM with Attention performed the best across the board, with the exception of Conferencing where LSTM was better by a slim margin. This suggests the attention mechanism is useful in identifying which parts of the packet sequence are learnable.

*Action recognition results* Our models outperform the baseline for most datasets, with accuracy improvements up to 26.31% more than that of the baseline for UCF101. For UCF101, we report test accuracy of 27.30%—the baseline action recognition model in the original paper [40] achieved 44.5% in 2012 when trained on the *unencrypted* data. While performance on the unencrypted data is likely to be much higher than 44.5% with current ML techniques, it is striking that we are able to achieve more than half of the original performance on encrypted data. We hypothesize the high performance is due to a combination of (a) the dataset’s higher-than-average resolution and (b) the diversity of movement across the classes (e.g., soccer juggling vs. playing cello). This diversity would manifest as different signatures in the packet sequences, making it easier to classify. Higher resolution data may have the effect of emphasizing the signature.

Next, we describe two methods of boosting training instances that slightly improved the advantage measure for some datasets:

1. One approach is to create *maximized* variants of certain datasets. Let  $N_c$  denote the number of training instances for class  $c \in C$ , where  $C$  is the set of all classes in a given dataset. Without maximizing, we balance the training set by taking  $\min_{c \in C} N_c$  training instances from each class. With maximizing, we take a subset  $D^* \subseteq C$  of classes such that  $D^*$  maximizes  $|D| \cdot \min_{d \in D} N_d$  over all possible subsets  $D \subseteq C$ . That is,  $D^*$  is constructed by removing classes from  $C$  until the total number of training instances in the balanced dataset is maximized over all possible subsets  $D \subseteq C$ .
2. We also saw advantage improvements after simplifying class labels based on class similarity. For example, the S-S dataset has many classes of the form “Holding X...” and “Holding Y...”—we truncate these classes to their root verb, in this case “Holding.”

We observe negligible performance gain for the Charades and S-S datasets. We hypothesize this low performance is attributed to two key factors:

1. These datasets are highly unbalanced to begin with. They have a lot of classes and high variation between the number of samples for each class. However, with some boosting of these datasets (maximizing and relabeling), performance gain improves to a degree—this suggests there may be a lack of data. Boosting improved performance the most with the LSTM with Attention model. After maximizing Charades and S-S, we measure advantage improvements from 0.52% to 1.60% for the former, and 0.32% to 3.52% for the latter. After relabeling S-S, we see an advantage improvement from 0.32% to 2.99%. After maximizing the relabeled S-S dataset, we see an even

larger advantage increase: from the aforementioned 0.32% (for pure dataset), 3.52% (maximized only), and 2.99% (re-labeled only) , to the maximized, re-labeled dataset’s 10.80%. As expected, we find that more data leads to better performance and believe that further improvements are tractable.

2. These datasets consist of highly similar videos: whereas the UCF101 and HMDB51 datasets have highly varied movements and backgrounds (e.g., as mentioned before for UCF101, soccer juggling vs. playing cello), the Charades and S-S datasets have near-identical movements with a static background (e.g., “Holding something next to something” vs. “Holding something in front of something”). We are able to learn when there are distinguishable burst patterns for each class; this holds for UCF101, HMDB51, and our Zoom conferencing dataset. On the other hand, it is difficult to learn when the classes are too similar.

*Zoom conferencing results* Our custom Conferencing dataset had the lowest number of distinct classes (3), and we were able to achieve test accuracy of 65.23% (advantage of 31.90% over random) from 67 epochs of training. As mentioned previously, this relatively strong performance can be attributed to the distinct burst patterns across the screen sharing, single participant, and gallery view classes. We also had 438.12 hours of Zoom conferencing data for three classes at our disposal—the most hours across the least classes among all our datasets. We believe model performance can be improved with a full emulation (See [Section 3.2](#)).

## 6 Related Work

We provide an overview of audio- and video-based fingerprinting in addition to relevant cryptographic works.

### 6.1 Fingerprinting

Fingerprinting tasks generally involve two stages: First, a dataset is constructed from label-trace pairs of known data. Second, this dataset is leveraged by a model to map labels in new packet traces (not contained in the original dataset) to the known dataset. This is effectively a constrained form of ML classification. Importantly, fingerprinting is designed to identify new traces to *known* labels (e.g., a website), not to perform predictive tasks. We section fingerprinting literature into a number of key subareas: audio-stream, voice-command, video-stream, and website fingerprinting (WF).

*Audio stream fingerprinting* Wright et al. [55] showed in 2007 that VBR-encoded VoIP traffic leaks the language of the underlying plaintext. Their classifier leverages packet lengths and was able to extract a range of language-based information from encrypted traffic. In subsequent work, Wright et al. [54] further showed that VBR-encoded VoIP traffic directly leaks fragments of the plaintext.

They were able to identify specific phrases from a standard speech corpus (i.e., a closed set of phrases) with an average accuracy of 50%, but greater than 90% for certain phrases.

White et al. [51] extended the phrase identification techniques of Wright et al. [54], showing that we can extract approximate transcripts without making closed set assumptions. They take a bottom-up approach, first identifying specific phonemes, then reconstructing words, and finally reconstructing sentences. Note their approach only works for the VoIP configuration analyzed in their paper, namely speech-specific VBR-encoded audio over SRTP.

Backes et al. [2] showed how to recover the identity of speakers participating in encrypted voice communication. They exploited the fingerprint produced by certain speakers and the times at which their voice would activate. In essence, speaker-specific speech patterns were reflected in how often their voice activated for a given voice activity detection (VAD) algorithm. This work again relies on speech-specific codecs and makes closed set assumptions. In particular, they analyzed the output of Speex (the codec) directly and did not simulate the encryption step that would happen in practice. They addressed the latter point by stating that LPE preserves length except for a constant offset, so the analysis of plain packets is equivalent to that of real traffic. In practice, their claim may not always hold: for example, the encryption algorithm may pad short inputs up to a minimum length.

*Video stream fingerprinting* Schuster, Shmatikov, and Tromer [36] were able to fingerprint video footage over a range of DASH streaming platforms. They observed that the bitrate produced by DASH when streaming a video encoded activity in source data, i.e., a “burst pattern.” The burst pattern of a video was found to be a strong fingerprint, meaning ML models can be trained to recognize a video’s fingerprint “in the wild” with reasonable robustness against different streaming platforms and environmental variance. The authors made heavy use of CNNs in their models.

In a similar pursuit, Gu et al. [18] also proposed a video identification method for network streaming over DASH traffic. The key difference in their approach was using a bespoke, bitrate-based feature extraction technique. This added robustness to their models.

Li et al. [27] took video fingerprinting a step further, showing that it is possible over sniffed Wi-Fi traffic (i.e., packets obtained “from the air”) as opposed to packets obtained directly at the IP layer. They used MLP- and RNN-based models that could identify streamed YouTube videos from a closed set. They compared their results to prior on-the-wire attacks that used CNNs, showing performance was roughly the same with about three-times-lower computation power and time.

More recently, Bae et al. [3] studied video identification attacks in the context of HAS over Long Term Evolution (LTE) networks. They showed an unprivileged device with the ability to broadcast radio signals can identify, with accuracy up to 98.5%, that a mobile user is watching a known video. Their attack relies on

CNNs trained on features extracted from LTE traces, so it is highly coupled to LTE network architecture.

## 6.2 Cryptography

Tezcan and Vaudenay [41] showed that in general, hiding message lengths is impossible for arbitrary message distributions. They proved that an exponentially-long padding is needed to ensure a negligible distinguisher advantage.

LHE attempts to overcome this impossibility. Paterson, Ristenpart, and Shrimpton [33] introduce a length-hiding parameter that is specified at encryption time. Naturally, it determines the amount of padding to be used for the encryption step. LHE is considered secure if for challenge messages  $m_0$  and  $m_1$ , it holds that  $0 \leq ||m_0| - |m_1|| \leq \Delta$  where  $\Delta$  depends on the length-hiding parameter.

Importantly, LHE does not solve the problem of how to choose the length-hiding parameter in practice. Gellert et al. [13] explore the possibility of automatically updating the length-hiding parameter in LHE. They proposed new security definitions to capture the leakage through message lengths. Their definitions allowed them to quantify the effectiveness of various countermeasures (such as padding) for specific plaintext distributions. They found that even a 2-5% bandwidth overhead from padding significantly reduced the effectiveness of fingerprinting attacks for known message distributions.

## 7 Conclusion

We empirically study leakage in the RTC context by measuring the learnability of neural networks on encrypted packets. In our results, we observe a spectrum upon which some video sources are more learnable than others: those that have well-defined classes with clear differences (burst patterns) lead to more measurable leakage. In these cases (HMDB51, UCF101, S-S relabeled and maximized, and Conferencing), our models outperform random by a non-negligible advantage. On the other hand, we were unable to learn on the datasets with highly similar classes (Charades, S-S).

**Disclosure of Interests.** We do not believe there is an urgent need for changes to Zoom or other platforms. However, our work is important to consider for future design choices in RTC. We have notified Zoom of our findings and will assist however needed.

## References

1. Ahmed, N., Natarajan, T., Rao, K.R.: Discrete cosine transform. *IEEE transactions on Computers* **100**(1), 90–93 (1974)
2. Backes, M., Doychev, G., Dürmuth, M., Köpf, B.: Speaker recognition in encrypted voice streams. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) *Computer Security – ESORICS 2010*. pp. 508–523. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

3. Bae, S., Son, M., Kim, D., Park, C., Lee, J., Son, S., Kim, Y.: Watching the watchers: Practical video identification attack in LTE networks. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 1307–1324. USENIX Association, Boston, MA (Aug 2022)
4. Cisco: Zero-trust security for webex. <https://www.cisco.com/c/en/us/solutions/collateral/collaboration/white-paper-c11-744553.pdf> (2021), online; accessed 19 October 2022
5. Corretgé, S.I., Ivov, E.: End-to-end encryption in jitsi meet. <https://jitsi.org/wp-content/uploads/2021/08/jitsi-e2ee-1.0.pdf> (2021), online; accessed 19 October 2022
6. Daemen, J., Rijmen, V.: Reijndael: The advanced encryption standard. *Dr. Dobb's Journal: Software Tools for the Professional Programmer* **26**(3), 137–139 (2001)
7. Défossez, A., Copet, J., Synnaeve, G., Adi, Y.: High fidelity neural audio compression. arXiv preprint arXiv:2210.13438 (2022)
8. Dodis, Y., Jost, D., Kesavan, B., Marcedone, A.: End-to-end encrypted zoom meetings: Proving security and strengthening liveness. In: Hazay, C., Stam, M. (eds.) *Advances in Cryptology – EUROCRYPT 2023*. pp. 157–189. Springer Nature Switzerland, Cham (2023)
9. Force, I.E.T.: Datagram transport layer security (dtls) extension to establish keys for the secure real-time transport protocol (srtp). <https://datatracker.ietf.org/doc/html/rfc5764> (2010), online; accessed 3 May 2023
10. Force, I.E.T.: Datagram transport layer security version 1.2. <https://www.rfc-editor.org/rfc/rfc6347> (2012), online; accessed 3 May 2023
11. Foundation, S.: ringrtc. <https://github.com/signalapp/ringrtc> (2022), online; accessed 19 October 2022
12. Garcia, K.D., de Sá, C.R., Poel, M., Carvalho, T., Mendes-Moreira, J., Cardoso, J.M., de Carvalho, A.C., Kok, J.N.: An ensemble of autonomous auto-encoders for human activity recognition. *Neurocomputing* **439**, 271–280 (2021)
13. Gellert, K., Jager, T., Lyu, L., Neuschulten, T.: On fingerprinting attacks and length-hiding encryption. In: Galbraith, S.D. (ed.) *Topics in Cryptology – CT-RSA 2022*. pp. 345–369. Springer International Publishing, Cham (2022)
14. Google: Google meet security & privacy for users. <https://support.google.com/meet/answer/9852160?hl=en#zippy=%2Cencryption> (2022), online; accessed 19 October 2022
15. Goyal, R., Ebrahimi Kahou, S., Michalski, V., Materzynska, J., Westphal, S., Kim, H., Haenel, V., Freund, I., Yianilos, P., Mueller-Freitag, M., et al.: The "something something" video database for learning and evaluating visual common sense. In: *Proceedings of the IEEE international conference on computer vision*. pp. 5842–5850 (2017)
16. Group, I.N.W.: Rtp: A transport protocol for real-time applications. <https://www.ietf.org/rfc/rfc3550.txt> (2003), online; accessed 3 May 2023
17. Group, I.N.W.: The secure real-time transport protocol (srtp). <https://www.rfc-editor.org/rfc/rfc3711> (2004), online; accessed 3 May 2023
18. Gu, J., Wang, J., Yu, Z., Shen, K.: Walls have ears: Traffic-based side-channel attack in video streaming. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. pp. 1538–1546 (2018)
19. Hasselquist, D., Vestlund, C., Johansson, N., Carlsson, N.: Twitch chat fingerprinting. In: Ensafi, R., Lutu, A., Sperotto, A., van Rijswijk-Deij, R. (eds.) *6th Network Traffic Measurement and Analysis Conference, TMA 2022, Enschede, The Netherlands, June 27-30, 2022*. IFIP (2022)

20. Isobe, T., Ito, R., Minematsu, K.: Security analysis of sframe. In: Bertino, E., Shulman, H., Waidner, M. (eds.) *Computer Security – ESORICS 2021*. pp. 127–146. Springer International Publishing, Cham (2021)
21. Juarez, M., Imani, M., Perry, M., Diaz, C., Wright, M.: Toward an efficient website fingerprinting defense. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) *Computer Security – ESORICS 2016*. pp. 27–46. Springer International Publishing, Cham (2016)
22. Kabal, P.: Wave file specifications. <https://www.mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html> (2022), online; accessed 30 August 2023
23. Kennedy, S., Li, H., Wang, C., Liu, H., Wang, B., Sun, W.: I can hear your alexa: Voice command fingerprinting on smart home speakers. In: *2019 IEEE Conference on Communications and Network Security (CNS)*. pp. 232–240 (2019)
24. Kleijn, W.B., Storus, A., Chinen, M., Denton, T., Lim, F.S., Luebs, A., Skoglund, J., Yeh, H.: Generative speech coding with predictive variance regularization. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. pp. 6478–6482. IEEE (2021)
25. Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., Serre, T.: Hmdb: a large video database for human motion recognition. In: *2011 International conference on computer vision*. pp. 2556–2563. IEEE (2011)
26. Ladune, T., Philippe, P.: Aivc: Artificial intelligence based video codec. In: *2022 IEEE International Conference on Image Processing (ICIP)*. pp. 316–320. IEEE (2022)
27. Li, Y., Huang, Y., Xu, R., Seneviratne, S., Thilakarathna, K., Cheng, A., Webb, D., Jourjon, G.: Deep content: Unveiling video streaming content from encrypted wifi traffic. In: *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. pp. 1–8 (2018)
28. Microsoft: Security and microsoft teams. <https://learn.microsoft.com/en-us/microsoftteams/teams-security-guide> (2022), online; accessed 19 October 2022
29. Microsoft: Satin: Microsoft’s latest ai-powered audio codec for real-time communications. <https://techcommunity.microsoft.com/t5/microsoft-teams-blog/satin-microsoft-s-latest-ai-powered-audio-codec-for-real-time/ba-p/2141382> (2023), online; accessed 3 May 2023
30. Nasr, M., Houmansadr, A., Mazumdar, A.: Compressive traffic analysis: A new paradigm for scalable traffic analysis. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. p. 2053–2069. CCS ’17, Association for Computing Machinery, New York, NY, USA (2017)
31. Omara, E., Uberti, J., Gouaillard, A., Murillo, S.G.: Secure frame (sframe). <https://datatracker.ietf.org/doc/draft-omara-sframe/> (2022), online; accessed 26 January 2023
32. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* **32** (2019)
33. Paterson, K.G., Ristenpart, T., Shrimpton, T.: Tag size does matter: Attacks and proofs for the tls record protocol. In: *Advances in Cryptology–ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security*, Seoul, South Korea, December 4-8, 2011. *Proceedings* 17. pp. 372–389. Springer (2011)
34. Project, W.: Vp9 video codec. <https://www.webmproject.org/vp9/> (2023), online; accessed 4 May 2023

35. Richardson, I.E.: The H. 264 advanced video compression standard. John Wiley & Sons (2011)
36. Schuster, R., Shmatikov, V., Tromer, E.: Beauty and the burst: Remote identification of encrypted video streams. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 1357–1374 (2017)
37. Sigurdsson, G.A., Varol, G., Wang, X., Farhadi, A., Laptev, I., Gupta, A.: Hollywood in homes: Crowdsourcing data collection for activity understanding. In: Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14. pp. 510–526. Springer (2016)
38. Sirinam, P., Imani, M., Juarez, M., Wright, M.: Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 1928–1943. CCS ’18, Association for Computing Machinery, New York, NY, USA (2018)
39. Software, O.B.: Obs studio. <https://obsproject.com/> (2023), online; accessed 4 May 2023
40. Soomro, K., Zamir, A.R., Shah, M.: Ucf101: A dataset of 101 human actions classes from videos in the wild. arXiv preprint arXiv:1212.0402 (2012)
41. Tezcan, C., Vaudenay, S.: On hiding a plaintext length by preencryption. In: ACNS. vol. 11, pp. 345–358. Springer (2011)
42. Thatcher, P.: How to build large-scale end-to-end encrypted group video calls. <https://signal.org/blog/how-to-build-encrypted-group-calls/> (2021), online; accessed 19 October 2022
43. Tomar, S.: Converting video formats with ffmpeg. Linux journal **2006**(146), 10 (2006)
44. Tudor, C.: The impact of the covid-19 pandemic on the global web and video conferencing saas market. Electronics **11**(16), 2633 (2022)
45. Valin, J.M., Vos, K., Terriberry, T.: Definition of the opus audio codec. Tech. rep. (2012)
46. Vass, J.: How discord handles two and half million concurrent voice users using webrtc. <https://discord.com/blog/how-discord-handles-two-and-half-million-concurrent-voice-users-using-webrtc> (2018), online; accessed 19 October 2022
47. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
48. Wang, C., Kennedy, S., Li, H., Hudson, K., Atluri, G., Wei, X., Sun, W., Wang, B.: Fingerprinting encrypted voice traffic on smart speakers with deep learning. In: Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks. p. 254–265. WiSec ’20, Association for Computing Machinery, New York, NY, USA (2020)
49. Wang, T., Goldberg, I.: Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In: Proceedings of the 26th USENIX Conference on Security Symposium. p. 1375–1390. SEC’17, USENIX Association, USA (2017)
50. WebRTC: Webrtc: Real-time communication in browsers. <https://www.w3.org/TR/2023/REC-webrtc-20230306/> (2023), online; accessed 3 May 2023
51. White, A.M., Matthews, A.R., Snow, K.Z., Monroe, F.: Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks. In: 2011 IEEE Symposium on Security and Privacy. pp. 3–18 (2011)
52. Wire: Wire security whitepaper. <https://wire-docs.wire.com/download/Wire+Security+Whitepaper.pdf> (2021), online; accessed 19 October 2022

53. Wireshark: The world's most popular network protocol analyzer. <https://www.wireshark.org/> (2023), online; accessed 4 May 2023
54. Wright, C.V., Ballard, L., Coull, S.E., Monrose, F., Masson, G.M.: Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In: 2008 IEEE Symposium on Security and Privacy (sp 2008). pp. 35–49 (2008)
55. Wright, C.V., Ballard, L., Monrose, F., Masson, G.M.: Language identification of encrypted VoIP traffic: Alejandra y roberto or alice and bob? In: 16th USENIX Security Symposium (USENIX Security 07). USENIX Association, Boston, MA (Aug 2007)
56. Zeghidour, N., Luebs, A., Omran, A., Skoglund, J., Tagliasacchi, M.: Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* **30**, 495–507 (2021)
57. Zhang, R., Hu, X., Li, B., Huang, S., Deng, H., Qiao, Y., Gao, P., Li, H.: Prompt, generate, then cache: Cascade of foundation models makes strong few-shot learners. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 15211–15222 (2023)
58. Zoom: Zoom cryptography whitepaper. [https://raw.githubusercontent.com/zoom/zoom-e2e-whitepaper/master/zoom\\_e2e.pdf](https://raw.githubusercontent.com/zoom/zoom-e2e-whitepaper/master/zoom_e2e.pdf) (2022), online; accessed 24 January 2023

# Supplementary Material

## 2 Extended Background

### 2.1 Multimedia Codecs

The choice of audio or video codec for use in (Web)RTC has powerful consequences for both security and call quality. Thus, the choice of codec (and the configuration with which it runs) depends on application requirements and bandwidth limitations per platform. Modern codecs, both in the audio and video settings, have one unifying feature: they are all lossily compressing. This means the source media is transformed such that information is lost about the source, but it can still be decoded with minimal human-perceptible audio or video degradation. For lossy compression to work effectively, the multimedia codec must be chosen carefully to ensure a reasonable trade-off between bandwidth cost and data quality. In general, the codecs exploit spatial and temporal redundancies in the source. For example, in a video recording, often a fraction of the background is static—the video codec can use this temporal redundancy to encode the background more efficiently. Similarly, if much of a video frame is a repeating pattern (spatial redundancy), the codec can encode the pattern once, then subsequently specify where the pattern appears in the frame, as opposed to encoding the entire frame naively.

While lossy compression is the main breakthrough that enabled RTC (via the Discrete Cosine Transform [1]), it magnifies existing leakage through packet lengths. Since data is encrypted “as it comes,” lossy compression exaggerates changes in a stream’s bitrate if the bitrate is allowed to vary. This increases the likelihood of leaking “burst patterns” [36] which encode information about the underlying data in the size of the encrypted packets. This data can be extracted with various techniques—we discuss prior work of this form in Section 6.

### 2.2 Protocols for Real-time Communication

In this work, we treat each RTC platform under evaluation as a black-box. While we do not manipulate the underlying RTC protocols directly, they play a part in the results and analysis. We give an overview of the key technical components within the investigated RTC platforms (see Table 1 for details) that influence leakage through encrypted packets. If the reader is familiar with (Web)RTC technologies, feel free to skip this section. The overview below is enough to read this paper; however, much detail is omitted—we refer the reader to the source material as cited.

*Web Real-time Communication (WebRTC)* Web Real-time Communication (WebRTC) [50] is an open-source project that provides simple APIs enabling RTC. In a nutshell, WebRTC is used to communicate data (usually, but not limited

to, audio and video) directly between web-enabled devices without proprietary software or external support, such as a server. It is designed to provide high-quality data streams over potentially low-bandwidth networks, and thus, makes heavy use of lossy codecs to compress data. WebRTC uses an array of standardized protocols to enable seamless point-to-point web communication. Notably, it uses the Secure Real-time Transport Protocol (SRTP) for encrypted data transmission and Session Traversal Utilities for NAT (STUN) to traverse network address translator (NAT) gateways for protocol establishment. For this paper, it is sufficient to focus on SRTP.

*Real-time Transport Protocol (RTP)* Real-time Transport Protocol (RTP) [16] is a communication protocol for delivering data over IP networks. Unlike its secure variant SRTP, it does not provide data confidentiality. RTP typically runs over UDP and works together with the RTP Control Protocol (RTCP). RTP carries the data streams (i.e., the audio and video), while RTCP monitors various transmission statistics and aids synchronization of multiple streams. RTP provides jitter compensation and handles lost or out-of-order packets—this helps retain robustness over UDP. It also allows data transfer to multiple destinations through IP multicast.

*Secure Real-time Transport Protocol (SRTP)* Secure Real-time Transport Protocol (SRTP) [17] is a profile for RTP which adds message confidentiality, integrity, and authentication. Like RTP, SRTP has a sister protocol called Secure RTCP (SRTCP) that serves the same purpose as RTCP for RTP. Naturally, SRTP gains security by encrypting the data payload with AES in counter mode for IP communications or f8 mode for telecommunications.

*Datagram Transport Layer Security (DTLS)* Datagram Transport Layer Security (DTLS) [10] is a channel security protocol that enables key management, parameter negotiation, and secure data transfer. It provides secure communication over unreliable transport protocols such as UDP. (It is analogous to regular TLS for TCP packets.)

*DTLS-SRTP* DTLS-SRTP [9] is a communication protocol for datagram-based applications. It fixes DTLS as the secure handshake protocol and SRTP as the secure data transmission protocol, allowing for fine-tuned protocol operation. Importantly, DTLS-SRTP can provide End-to-end Encryption (E2EE) [8]—a property that cannot be realized with the subprotocols independently.

*Secure Frame (SFrame)* Secure Frame (SFrame) [31] is a (group) communication protocol for RTC that provides E2EE. It assumes that many clients interact with a central server through which they can communicate with each other. Each point-to-point connection (i.e., each client connection with the server) is protected by DTLS-SRTP. In SFrame, each client has a public key pair—SFrame does not enforce how these keys are distributed. Importantly, clients first encrypt RTP audio and video frames, then the encrypted frame is packetized. The

packetization process adds various metadata to enable decryption on the other end.

Variants of SFrame have been deployed to a number of RTC platforms in practice. In particular, Jitsi Meet [5], Signal [11], and Cisco WebEx [4] use custom versions of the protocol that are tailored to their platform. In 2021, Isobe, Ito, and Minematsu [20] formally analyzed SFrame. They found that the original specification allowed for an impersonation attack, which resulted in an update to the specification draft by SFrame’s working group—they removed the signature mechanism that permitted the attack and planned to support it in the future.

### 3 Extended Related Work

#### 3.1 Voice Command Fingerprinting

Kennedy et al. [23] demonstrated a voice command fingerprinting attack on smart home speakers. Given the packets traveling between the home device and a cloud server, an attacker can infer a user’s voice commands. They introduce the notion of semantic distance, which measures privacy leakage with respect to natural language processing. Finally, they showed that heavy padding significantly lowers classification performance, though it comes at a high bandwidth cost.

Wang et al. [48] also used bidirectional network metadata between smart speakers and cloud servers to infer voice commands. They achieve high accuracy using DL. Their attack is effective because the server responds deterministically, leaving a distinguishable pattern over encrypted traffic. The authors also built an automated traffic collection tool and proposed a defense mechanism that mitigates their models.

#### 3.2 Website Fingerprinting

Nasr, Houmansadr and Mazumdar [30] introduced the notion of compressive traffic analysis, in which traffic features are compressed such that models are not trained on raw traffic features. Instead, they are trained on a more efficient representation of data, saving storage, bandwidth, and computation along the way. The authors use techniques from signal processing, such as linear projection algorithms, to do the compression. The main improvement over prior work is computational efficiency, meaning that WF and related traffic analysis tasks can be performed at much wider scale than allowed by previous techniques.

Payap et al. [38] introduced Deep Fingerprinting (DF): a CNN-based fingerprinting attack on Tor. DF was able to defeat prior fingerprinting defense mechanisms such as WTF-PAD[21] (90% accuracy), though Walkie-Talkie[49] was more effective against DF (49.7% accuracy). Moreover, they achieve over 98% accuracy on Tor traffic *without* defenses and even higher metrics on non-Tor traffic. All accuracy numbers are with respect to known websites.

## 4 Experimental Parameters

*Devices* We make use of the following devices for packet capturing and model training: an array of basic laptop machines to perform automated data collection, and a powerful (albeit consumer-grade) desktop machine used for model training. The hardware specifications are as follows:

1. **Machine A.** A Linux laptop with an 8th generation Intel Core i9, NVIDIA GeForce GTX 1050 Ti Max-Q, and 32 GiB of DDR4 RAM.
2. **Machine B.** A dual-booted Windows/Linux laptop with an 8th generation Intel Core i7, no discrete graphics, and 8 GiB of DDR4 RAM.
3. **Machines C and D.** Two identical Linux laptops with an 8th generation Intel Core i7, no discrete graphics, and 16 GiB of DDR4 RAM.
4. **Training machine.** Training machine running Linux with a 12th generation Intel Core i9 CPU, NVIDIA GeForce RTX 3090 Ti GPU, and 32 GiB of DDR5 RAM.

Machines A and C were designated as *streamer* devices, and Machines B and D were designated as *capturer* devices. Machine A was paired with C, and B was paired with D. For synthetic data specifically, we found it was necessary for the streaming machine to have dedicated graphics capabilities in order to encode high entropy media without introducing severe latency. Hence, extreme audio and video was only evaluated using the A-B machine pair with discrete graphics enabled. All non-synthetic datasets were evaluated with discrete graphics disabled on Machine A to achieve consistency with Machines B, C, and D.

*Network* All data was collected over Wi-Fi networks (802.11ac and newer) capable of at most 1Gbps upload and download speeds. This ensured there was no bottleneck to RTC platforms.

## 5 Dataset Metadata

*Preprocessing* We re-encoded all datasets with the H.264 codec [35] to scale footage to 720p—this is a globally supported resolution for webcam footage. This was done with the following FFmpeg [43] configuration: `ffmpeg -i {input} -vf scale=1280:720 -preset slow -crf 18 {output}`.

## 6 AI Codecs

Lyra is able to achieve wideband quality at 3kbps and Satin achieves super-wideband quality at 8kbps. For comparison, Opus reaches narrowband quality at about 10 kbps. This efficiency comes at the cost of performance: it is much more CPU-intensive to perform AI-enhanced encoding and decoding, resulting in higher latency. Moreover, AI codecs introduce significant metadata overhead to allow decoding on the other end. In some cases, a header can even dwarf the payload, as was the case in early versions of Lyra.

Dataset	Description	Hours	Recordings	Traces
<i>Synthetic Audio</i>	Extreme synthetic audio	0.2	6	18
<i>Synthetic Video</i>	Extreme synthetic video	0.2	6	18
<i>HMDB51</i>	Human motion clips	5.98	6,766	7,470
<i>UCF101</i>	Realistic, diverse human action clips	26.7	13,320	13,407
<i>Charades</i>	Daily indoor activities	81.36	9,848	10,031
<i>Something-Something</i>	Object manipulation	234.24	220,847	236,302
<i>Custom Zoom</i>	Public Zoom meeting recordings	438.12	1,146	1,146
Total:		786.8	251,939	268,392

**Table 3.** Overview of all datasets broken down by number of hours of footage, number of distinct recordings, and number of network traces captured. The number of traces is at least as high as the number of recordings—it was sometimes necessary to re-run data collection due to a network outage or a runtime error.

	Filename	File Size (KB)		Filename	File Size (KB)
<i>Audio</i>	U	20,673	<i>Video</i>	U	4,441,283
	W	10,337		W	171
	UW	20,673		UW	2,220,746
	WU	20,673		WU	2,220,746
	UWUW	20,673		UWUW	2,220,778
	WUWU	20,673		WUWU	2,220,778

**Table 4.** File sizes for synthetic high (U) and low (W) entropy media and permutations measured in kilobytes. Observe that  $|UW| = |WU|$  and  $|UWUW| = |WUWU|$ . All audio files were encoded with lossless WAV, and all video files were encoded with lossless VP9.

Satin is not a standalone codec like Lyra—Microsoft instead opted to extend Opus using ML techniques. Consequently, Satin supports both low- and high-bitrate modes and can switch between them on-the-fly. For a taste of Satin’s performance, Silk cannot produce a signal for audio below 4kHz at 6kbps, whereas Satin can produce signals up to 16kHz at the same bitrate—this quality improvement is comparable to that of the transition from G.711 to Opus. Most importantly, Satin is polished enough for real systems (e.g. it did not suffer from the latency problems of Lyra V1), and it is deployed at scale in Microsoft Teams as of 2021.

## 7 Concerns and Recommendations

*Analytics over encrypted RTC* In this paper, we work on a small scale in the grand scheme of things—adversarial entities could see better model performance just by leveraging more data than we have. We show that with data obtainable in about 800 hours with a pair of computers, DL models can learn non-trivial information from encrypted packets. The bottleneck to model performance is collecting labeled data for training: to learn from encrypted data, an adversary must have a large amount of data distributed similarly to the plaintext producing the packets from which she wishes to learn. There are entities with potentially anti-consumer interests that may be able to exploit their position: RTC providers

themselves and Wi-Fi router manufacturers. These entities have a financial incentive to learn over masses of encrypted packets—one straightforward incentive is to collect data for targeted advertising. Our results give strong evidence that probabilistically extracting information over encrypted packets is feasible in the real world. This would allow the aforementioned entities to (a) claim that their systems are end-to-end encrypted, yet (b) retain the ability to perform simple analytics over encryption.

*VBR vs. CBR* Our models are effective because bitrates are allowed to vary. That is, RTC platforms have opted for the benefits of VBR (namely, lower bandwidth) over the benefits of data-hungry alternatives like CBR (more stable streams, less leakage). The usage of VBR codecs with LPE has actually grown among the largest RTC providers (Zoom [58], Google Meet [14], Microsoft Teams [28]) This paper adds to the decade-long line of work demonstrating that VBR is potentially exploitable, and thus we challenge the current VBR paradigm. We make the following remarks:

1. CBR is currently deployed in at least two RTC platforms: Signal [11] (default behavior) and Wire [52] (as an option). In writing this paper, we performed small-scale testing on Signal and found it was less susceptible to leakage: it would be significantly more difficult for a model to learn from encrypted packets produced by Signal. Moreover, Signal operates at relatively high scale and even supports group video calls, indicating CBR is not a barrier to scaling.
2. We suggest that switching from VBR to CBR is low-cost, both in terms of one-time development costs and ongoing bandwidth costs, in the current RTC climate. If bandwidth is limiting, one could envision a “two-stage” mechanism where if participants in a conference have sufficient network capabilities, CBR is used, and if not, the system falls back to VBR to preserve functionality at the cost of security.

*Future codecs: AI-powered compression* In 2021, Microsoft and Google both released their own AI-enhanced audio codecs, Satin [29] and Lyra [24], respectively. These codecs are designed to outperform Opus [45] and any other non-AI codec at low bitrate settings, i.e. they are designed to aggressively compress audio to achieve the highest quality possible with the lowest bitrate cost. See [Supplement 6](#) for further details about Satin and Lyra. More advanced AI codecs are in development, such as SoundStream [56] from Google and Encodec [7] from Meta. The video setting is also being explored [26].

We conjecture these AI codecs, if composed with encryption in the standard way, will inevitably lead to more leakage. Recall in [Section 4](#) that `teams` in the audio setting did not adapt well to the synthetic tests relative to apps that use Opus. Further study is necessary to evaluate AI codecs in the audio setting (both live, i.e. in Teams, and those in development), as well as those in the video setting.

## 8 Future Work

*Model optimization* There is room for technical optimization with respect to our model design. First, more powerful model architectures for sequential data, such as bi-directional LSTMs and Transformers [47], may yield better performance for our classification tasks, though these complex models likely require more training data and computation power. Second, we may be able to represent the training data in a more efficient form. At present, we train the neural nets with long, high-fidelity packet length sequences. One way to prune redundant data is to make use of the way bitrate changes for training data. We could analyze the points where the bitrate curve is steep for training data and label data at these specific points. That is, we can encode information at the precise points that are likely to have a strong imprint on encrypted packets. Compressed sensing techniques may also be applicable [30]. Third, we note that we only conducted a single hyperparameter sweep and used the same parameters for all trained models. However, the datasets differ substantially, and carrying out a hyperparameter sweep to optimize the parameters for each individual model and dataset may result in better performance.

*Model ensembles* In this work, we developed independent models that are trained to perform a specific task. However, these tasks are related, and one can imagine using multiple models to make deeper inferences on encrypted packets. This idea has seen success in both fingerprinting (e.g., [3]) and the activity recognition literature (e.g. [12]). One example could be to first determine if a meeting participant’s webcam is full-screened with one model, then, use a second model to determine what action he is performing. This extension follows directly from our paper’s results.

*Improved Conferencing data collection* While our study is the first to automatically collect real encrypted packets from RTC platforms, our automation has limitations (see Section 3.2). Most importantly, we stream data from the Streamer to the Capturer, which is faithful to server mixing RTC platforms. For platforms that use selective forwarding or full mesh architectures, it would be accurate to capture packets at both ends of the call and stream only the data that should come from one end to the other. For example, if reproducing a meeting recording of Alice and Bob conversing, the framework should only send data that Alice says to Bob (who then captures the relevant packets) and vice versa. This can be generalized to calls with more than two participants.

*Audio setting* This work focuses on the video setting (rather than audio) because our characteristic analysis suggested the video setting might be more susceptible to leakage. It may be worth revisiting the transcript extraction problem [51] to see if the results are reproducible with DL on live platforms. At present, DL is well-equipped to handle sequence-to-sequence learning (i.e., transformers [47]) and we believe that approximate transcript extraction is possible over encrypted packets—if demonstrated, this would defeat encryption.

*Theory* Our current security definitions do not suffice. While we know impossibility results exist [41], we can at least use LHE with a scientifically-determined length-hiding parameter [13]. New definitions might benefit from aiming to capture the length distributions produced by realistic message spaces. One idea is to map plaintext messages to tiers of message sizes—this effectively places packets in distinct groups where packets in a group are padded up to the group’s max size. In this case, for security to hold, messages should be indistinguishable such that  $0 \leq ||m_0| - |m_1|| \leq \Delta_{m_0, m_1}$ . Importantly, the group size  $\Delta_{m_0, m_1}$  is allowed to depend on the messages in question. A definition of this kind is likely to be in step with the security provided by CBR. Future theoretical work is necessary to design a natural and realistic security framework.