Iterative Text-based Editing of Talking-heads Using Neural Retargeting

XINWEI YAO, Stanford University
OHAD FRIED, The Interdisciplinary Center Herzliya
KAYVON FATAHALIAN, Stanford University
MANEESH AGRAWALA, Stanford University

Fig. 1. Our iterative text-based tool for editing talking-head video takes 2-3 minutes of a target video as input and is designed to support an iterative editing workflow. On each iteration the user might edit the wording of the speech (itr. 1 and 2), refine mouth motions if necessary to reduce artifacts, manipulate the performance by inserting mouth gestures (itr. N) or change the overall speaking style (e.g. energetic, mumble). Our tool requires only 2-3 minutes of the target actor video and it synthesizes the video for each iteration in about 40 seconds, allowing users to quickly explore many editing possibilities as they iterate. Our approach is based on two key ideas. (1) We develop a fast phoneme search algorithm that can quickly identify phoneme-level subsequences of the source repository video that best match a desired edit. This enables our fast iteration loop. (2) We leverage a large repository of video of a source actor and develop a new self-supervised neural retargeting technique for transferring the mouth motions of the source actor to the target actor. This allows us to work with relatively short target actor videos, making our approach applicable in many real-world editing scenarios. Finally, our refinement and performance controls give users the ability to further fine-tune the synthesized results.

CCS Concepts:
- Computing methodologies → Motion processing: Computational photography; Reconstruction; Graphics systems and interfaces.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM. 0730-0301/2021/ART1 $15.00 https://doi.org/10.1145/3449063

1 INTRODUCTION

Tools for editing talking-head video using transcripts have made it possible to easily remove filler words, emphasize phrases, correct mistakes, and try different wordings of the speech [Berthouzoz et al. 2012; Fried et al. 2019; Suwajanakorn et al. 2017; Thies et al. 2020]. Many of these tools can synthesize high-quality results that closely match the appearance of the unedited video. Such tools have the potential to enable a variety of post-capture editing applications including re-phrasing dialogue in a film scene, dubbing commercials to a new language, developing dialogue for a conversational video assistant, and fixing wording errors in an online lecture.

Yet, current transcript-based video editing tools are impractical for use in many real-world editing scenarios for four main reasons.

(1) **Slow feedback loop hinders iterative editing.** Synthesizing the edited result at high-quality is often extremely slow. For example, while viewers report that Fried et al.’s [2019] results appear very realistic, their approach takes hours to generate a few seconds of edited video. The slow feedback loop — time between specifying an edit and seeing the result — significantly hinders iterative editing (e.g. trying different phrasings of dialogue).

(2) **Require hours of target talking-head video.** To produce realistic results, many of these tools [Fried et al. 2019; Suwajanakorn...
et al. 2017] require hours of video of the target talking-head actor. Some tools further require the actor to speak a set of specialized phrases (e.g. TIMIT corpus [Garofolo et al. 1993]). In practice however, many video editing projects lack access to such large amounts of target actor video.

(3) **Missing controls for refining results.** None of these editing tools provide controls for manually refining the lip motions of the synthesized results, making it impossible to fix objectionable artifacts these editing tools sometimes generate (e.g. mouth doesn’t fully close on \m, \b, \p phonemes).

(4) **Missing controls for adjusting non-verbal performance.** None of these editing tools include controls for adjusting the target actor’s non-verbal performance by inserting mouth gestures (e.g. a smile) or changing the overall speaking style (e.g. mumbling, energetic).

In this work we present an iterative talking-head video editing tool that explicitly addresses all four of these issues. While our approach builds on the high-quality synthesis technique of Fried et al. [2019], we make several new contributions. We significantly reduce the time required to synthesize video (from hours to about 40 seconds for a 6 word edit) by developing a fast algorithm for searching the source repository for the desired lip motions. We lower the data requirement on the target actor video (2-3 minutes are usually enough) by leveraging a large repository of video from a source actor and use a new self-supervised neural retargeting technique to transfer their lip motions to the target actor. We provide controls to refine results by allowing users to smooth over jumpy transitions and force mouth closure on the results of the automated synthesis pipeline. Finally, we enable insertion of non-verbal mouth gestures with the same text interface, as well as controls to switch between different speaking styles by using a version of the source repository with the desired style.

As shown in Figure 1 our tool enables an iterative editing workflow. Given a short video of the target actor, the user can edit the transcript and our tool synthesizes the corresponding video in under a minute. The user can inspect the feedback and further adjust wording, refine the lip motions and/or insert mouth gestures and quickly see how the adjustment affects the synthesized video. Note that our work focuses on generating video from text; to obtain the corresponding speech audio, we rely on either having access to the actor speaking the new content (e.g. from a prerecorded library of the actor’s speech or recorded by the actor in real-time during editing), text-to-speech voice synthesis [van den Oord et al. 2016] or voice cloning [Jia et al. 2018; Kumar et al. 2019].

We demonstrate a variety of iterative editing sessions facilitated by our tool and we conduct user studies which show that our synthesized results are rated as “real” for 56.2% of the sentence-long edits and for 64.9% of the phrase-long edits — slightly better than the previous state-of-the-art approach of Fried et al. [2019]. Together these results suggest that our algorithm provides the speed, data efficiency and controls necessary for a practical iterative editing workflow while maintaining high-quality synthesis results.

2 RELATED WORK

**Video-driven talking-head synthesis.** A common approach to synthesizing a talking-head video is to use a “driving” video from a different actor that has the desired motion, expression and speech, and transfer those elements to the primary talking-head. Early attempts used facial landmarks from a video to retrieve frames of a different person and play them back directly [Kemelmacher-Shlizerman et al. 2010] or after warping [Garrido et al. 2014]. Opting for a lower data requirement, several approaches synthesize video given only one or a few photos of the target person, either by morphing and blending [Averbuch-Elor et al. 2017] or using neural networks [Geng et al. 2018; Pumarola et al. 2019; Wiles et al. 2018; Zakharov et al. 2019]. These methods are successful in producing short expression videos, but are less convincing for full sentences. Several approaches use a tracked head model, to decouple properties (e.g. pose, identity, expressions) to produce convincing results [Garrido et al. 2015; Kim et al. 2019, 2018; Thies et al. 2016; Vlasic et al. 2005]. We similarly use a tracked head model to decouple such properties. All of these previous methods require a driving video to specify the desired output head motion and expression. In this work we specify those properties via text, which is often a simpler, lower-cost interaction.

**Voice-driven talking-head synthesis.** Another approach for talking-head synthesis is to drive it with voice. The pioneering work of Bregler et al. [1997] creates talking-heads through a combination of alignment and blending, and was improved upon in various follow-ups [Chang and Ezzat 2005; Ezzat et al. 2002; Liu and Ostermann 2011]. Others have used human voice-driven synthesis to dub non-humans [Fried and Agrawala 2019]. Several methods synthesize a talking-head given only one or a few video frames in addition to the voice track [Chen et al. 2019; Chung et al. 2017; Song et al. 2019; Vougioukas et al. 2018, 2019; Zhou et al. 2019]. However, the result is a fixed frame with a moving inner-face region, or a tightly cropped head, and is easily distinguishable from realistic video. Suwajanakorn et al. [2017] demonstrate that using a large repository of video (17 hours) can produce convincing synthesis results. In work concurrent to ours, Thies et al. [2020] produce video from speech. We compare to their results in Section 5.3. None of these voice-drive methods provide refinement and performance controls that are essential for iterative editing.

**Text-driven talking-head synthesis.** Most related to our work are methods that perform text-based video editing and synthesis. Wang et al. [2011] synthesize a talking-head, and allow control over facial expressions, but the head is floating in space and is not part of a photorealistic video. Mattheyes et al. [2010] synthesize audio-visual speech from text, but the resulting videos have no head motion making them unrealistic. Berthouzoz et al. [2012] can edit talking-head video by cutting, copying and pasting transcript text. However, they do not allow synthesis of new words to change phrasing or fix flubbed lines. ObamaNet [Kumar et al. 2017] synthesizes both audio and video from text, using a large dataset of 17 hours of the president’s speeches. While predominantly an audio-based method, Thies et al. [2020] also show text-based results by incorporating a text-to-speech system. The work of Fried et al. [2019] most closely resembles ours, but it requires over 1 hour of target video and takes
hours to produce a result, while our tool requires 2–3 minutes of target video and produces results in about 40 seconds. We compare our results to both Thies et al. and Fried et al. in Sections 3.3 and 5.4 and find that the quality of our results is similar to both of these techniques. Moreover we provide refinement and performance controls that are missing in all previous text-driven talking-head synthesis tools, but are critical for a practical video editing tool.

3 METHOD

Given a short talking-head video of a target actor (often 2-3 minutes in length), and an edit of the video transcript, our system synthesizes new video of the target actor matching the edit. An edit is specified as a replacement of one continuous sequence of words in the original transcript with a new sequence of words. Since a short target actor video is unlikely to contain all the lip motions necessary to convincingly synthesize the sequence of phonemes in the edit, we leverage a large repository of video from a different, source actor. Specifically, we pre-capture an hour of a source actor speaking the TIMIT corpus [Garofolo et al. 1993] which includes the most common phoneme combinations (coarticulations) in English and we retrain their lip motions to the target actor during synthesis.

Our approach for quickly synthesizing the edited result is based on the approach of Fried et al. [2019] but involves several critical modifications. As in Fried et al., our preprocessing pipeline (Figure 2) annotates both the repository and target videos with phonemes and registers a parametric 3D head model to the face in each frame of each video. Our synthesis pipeline (Figure 3) provides a new, fast phoneme search algorithm that finds subsequences of phonemes in the source video that match the desired edit. It then stitches together the corresponding parameters of the 3D head model for the source actor across subsequence boundaries to smooth the lip motions. We introduce a new self-supervised neural retargeting step that adapts the parameters representing the lip motion of the source actor to those of the target actor and blend the resulting parameters into the target video. Finally we render photorealistic frames from the parameters using neural rendering [Tewari et al. 2020].

We briefly summarize how we adapt each step in Fried et al.’s pipelines to our problem in Sections 3.1 and 3.2. We then present the details of our new algorithms; fast phoneme search and stitching in Section 3.3 and neural retargeting algorithm in Section 3.4. In Section 3.5, we describe the iterative refinement and performance controls enabled by our approach.

3.1 Preprocessing Pipeline

Our preprocessing pipeline annotates each frame of the repository and the target videos in two main steps, (1) phoneme alignment and (2) registration of a parametric 3D head model. These resulting phoneme and face parameter annotations are used by our synthesis pipeline to establish correspondences between the target video and source repository. Note that the repository is only annotated once and the resulting annotated repository is then bundled as part of the system. In contrast, the target video must be annotated each time a new target video is given as input.

**Phoneme Alignment.** The phoneme alignment step takes as input a video (repository or target) paired with its text transcript, and computes the identity and timing of the phonemes in the video. Specifically, we use P2FA [Rubin et al. 2013; Yuan and Liberman 2008] to convert the transcript into phonemes and align them to the audio speech track of the video. This produces an ordered sequence $V = (v_1, ..., v_n)$ of phonemes, where each phoneme $v_i$ contains its name, start time and end time. If the transcript is not available, we can obtain one using a transcription service such as Google Cloud Speech-to-Text [2020a], or rev.com [2020].

**3D Head Model Registration.** We fit a parametric head model [Blanz and Vetter 1999; Thies et al. 2016] to each frame of video using a monocular head tracker [Garrido et al. 2016]. At every frame, the fitted model includes 80 parameters for 3D facial geometry, 80 for facial reflectance, 3 for head pose, 27 for scene illumination and 64 for face and lip expressions. In the fitting procedure we hold the facial geometry and reflectance parameters constant across all the frames of the same actor, but we allow the pose, illumination and expression to vary across time.

3.2 Synthesis Pipeline

Our synthesis technique is based on matching phonemes in the edit to phonemes in the repository. Therefore, we first convert the input text of the edit from words into a sequence of phonemes $\mathcal{W} = (w_1, ..., w_m)$, where each phoneme contains its name, start time and end time. Specifically, we convert the edit into audio using either text-to-speech voice synthesis [goo 2020b; van den Oord et al. 2016] or voice cloning techniques [lyr 2020; Kumar et al. 2019], and then apply P2FA [Rubin et al. 2013; Yuan and Liberman 2008] to time-align the resulting speech to the phonemes of the edit. Note that our synthesis algorithm only uses the timing of the phonemes and does not use any other aspect of the synthesized speech audio signal. If the user has access to the audio of the target actor saying the new content (either from a prerecorded library of the actor’s speech or recorded by the actor in real-time during editing) they can run our fast synthesis pipeline with the timings obtained from the real-voice recordings.

**Fast Phoneme Search and Stitching.** The fast phoneme search and stitching step is designed to quickly find the best subsequences of phonemes in the repository and then stitch together the corresponding expression parameters of the source actor 3D head model, in order to produce the the edit $\mathcal{W}$ we wish to synthesize. Our new algorithm operates three orders of magnitude faster than Fried et al. [2019] and finds the best repository subsequences in
seconds rather than hours. We present this fast algorithm in detail in Section 3.3.

**Neural Retargeting.** The retargeting step converts a sequence of expression parameters for the source actor into those for the target actor. We introduce a learned retargeting model, trained in a self-supervised manner from corresponding pairs of repository and target video sequences and transforms the expression parameters as detailed in Section 3.4. The result is a sequence of target actor face expression parameters corresponding to the edit.

**Expansion to Full Parameters.** Next, we combine the synthesized target actor expression parameters with geometry, reflectance, pose and illumination parameters from the input target video to produce a sequence of full face parameters for the target actor corresponding to the edit. Specifically, we take an interval of frames around the edit location in the target video, retime it to account for the duration of the edit, and use the geometry, reflectance, pose and illumination parameters from the retimed interval.

**Neural Rendering.** The neural rendering step takes the sequence of full face parameters for the target actor and first generates a composite image in which the lower face region is a rendering of the 3D head model, while the upper part of the head and the surrounding background are from the original target video, but retimed to match the length of the edit. It then uses a GAN trained on the target video to complete the image-to-image translation from composite image to photorealistic frame.

### 3.3 Fast Phoneme Search and Stitching

Our synthesis pipeline takes an edit \( W \) specified as a sequence of phonemes with timings \( (w_1, \ldots, w_m) \) and starts by finding matching subsequences of video in the source repository \( V \), that can be combined to produce \( W \). More precisely, we partition the edit \( W \) into phoneme subsequences \( (W_1, W_2, \ldots, W_k) \) and for each subsequence \( W_i \) find its best match \( V_i \) in the repository \( V \). Fried et al. [2019] use a brute-force method that considers all possible partitions \( \text{split}(W) \) of the edit \( W \), and all possible matches with subsequences of \( V \) to find \( (V_1, V_2, \ldots, V_k) \) that minimizes the objective:

\[
C(W, V) = \min_{(W_1, W_2, \ldots, W_k) \in \text{split}(W)} \sum_{i=1}^{k} C_{\text{match}}(W_i, V_i) + C_{\text{len}}(W_i)
\]

where \( C_{\text{match}} \) is a custom Levenshtein edit distance [1966] between two phoneme subsequences that takes into account the phoneme label, the viseme label and the timing difference, and \( C_{\text{len}} \) penalizes short subsequences. In order to find subsequences that transition well at their endpoints, during the search, we expand each subsequence \( W_i \) by a single phoneme on either end. Thus, adjacent subsequences overlap by two context phonemes. We find that this new context expansion approach better captures co-articulation effects between the subsequences, than the algorithm of Fried et al. which does not use context expansion (see user studies in Section 5.4).

We further modify Fried et al.’s search algorithm in three key ways to obtain a speedup of over 3 orders of magnitude: (1) we propose a fast alternative to the Levenshtein distance, (2) we reduce the size of the search space on \( W_i \) and, (3) given \( W_i \), we use a viseme-based indexing scheme to quickly find the optimal \( V_i \). Finally, we stitch together source actor expression parameters corresponding to the \( V_i \)‘s to produce a single coherent sequence of expression parameters.

(1) **Fast alternative to edit distance.** The full Levenshtein edit distance allows substitution, insertion and deletion of phonemes when computing \( C_{\text{match}} \). However, we have observed that when the matching subsequences between the edit \( W \) and the repository video \( V \) contain phoneme insertions or deletions, the final synthesized video appears out-of-sync with the audio; it either contains extraneous mouth motions due to phoneme insertion, or it misses mouth motions due to deletion. In practice we find it is beneficial to disallow insertions and deletions and only allow phoneme substitutions. Given a subsequence \( W_i \) of the edit \( W \), this approach forces \( C_{\text{match}} \) to only consider subsequences \( V_i \) of the repository \( V \) that contain the same number of phonemes as \( W_i \). We can therefore replace the Levenshtein distance with the sum of element-wise substitution...
cost which requires linear time in the number of phonemes rather than the quadratic time required for computing the full Levenshtein distance [Wagner and Fischer 1974].

(2) Reduce search space for partitioning. The brute force search considers all possible partitions of \( W \) into \( (W_1, \ldots, W_K) \). But, an extremely long edit subsequence \( W_i \) is unlikely to have a good match with a repository subsequence \( V_i \). Thus, we can reduce the search space of possible partitions by capping the maximum length of the \( W_i \)'s to \( L \). In our experience, 99% of the matches found by the brute force search are of length 6 or less, and we therefore set \( L = 6 \). This approach reduces the number of partitions to search. More importantly, it typically reduces the number of distinct \( W_i \)'s we need to consider by over an order of magnitude, especially when \( W \) the full edit sequence is itself very long.

(3) Viseme-based index to search repository. For each edit subsequence \( W_i \) we consider in our search space, we must find the optimal \( V_i \) in the repository with respect to \( C_{\text{match}} \). Instead of checking all possible subsequences in the repository, we impose an additional constraint on \( V_i \) that restricts the set of \( V_i \) we consider to only the most likely match candidates and allows us to build an index structure on the set of \( V_i \) to retrieve the likely candidates quickly.

As in Fried et al., our \( C_{\text{match}} \) cost function considers phonemes to match when they appear visually similar – that is, their corresponding visemes match. By imposing the restriction that \( V_i \) start with the same viseme n-gram as \( W_i \) we can pre-compute an index for the repository using viseme n-grams as the key and the location of the n-gram in the source repository video as the value. At search time, we look up all possible candidate \( V_i \)'s using this index and only compute the \( C_{\text{match}} \) for them. While this indexing approach speeds up the search significantly it also reduces the space of subsequence matches the search considers. In general, the longer the n-gram key, the stronger the reduction and the more likely it is that no good match will be found. In practice, we find that using a bi-gram index best balances this trade-off between search speed and result quality.

Stitching. After we find the best matching phoneme subsequences \( V_1, \ldots, V_k \) from the repository, we look up the expression parameters of the source actor’s 3D head model corresponding to each phoneme, and linearly re-time them to match the phoneme durations specified in the edit. We then stitch together adjacent subsequences by first linearly blending the expression parameters across the overlapping context phonemes and then applying a Gaussian filter over a window of 4 frames around the transition boundaries between the subsequences to further smooth the transition.

We have found in practice, that errors in tracking expression parameters, timing misalignments and our linear blending can sometimes fail to fully close the mouth of the 3D head model at the beginning of ‘m’, ‘b’, and ‘p’ phonemes. However, proper mouth closures for these phonemes is crucial for producing perceptually realistic results [Agarwal et al. 2020]. We therefore force the desired mouth closure by linearly blending in a closed-mouth expression from the repository at the beginning of all ‘m’, ‘b’, and ‘p’ phonemes with a default length of 2 frames. Note that the closed-mouth expressions are manually annotated once during repository preprocessing and

we automatically use the one closest to the parameter values at the point of insertion.

Overall, the fast search strategy reduce the runtime of the phoneme search process by three orders of magnitude compared to the brute-force approach. It takes around 5 seconds to find and stitch snippets \( V_1, \ldots, V_k \) for an edit \( W \) of 20 phonemes in an hour-long repository.

3.4 Neural Retargeting

Given a stitched-together sequence of face expression parameters for the source actor in the repository, the goal of retargeting is to generate a matching sequence of expression parameters for the target actor. We have developed a self-supervised neural network model for retargeting and in Section 5.1 we show that using a neural network for retargeting produces higher-quality results than baseline methods such as directly copying source actor expression parameters to the target actor, or applying a linear retargeting model.

Self-supervised training data. To train our retargeting network we require sequences of expression parameters for the source and target actor that correspond to one another with respect to their mouth motions. Assuming that uttering the same sequence of visemes will produce similar mouth motions, we automatically construct corresponding pairs of training data by finding the longest matching sequences of phonemes between the source repository video and the target video, as follows.

Since we apply our retargeting model to a stitched-together sub-sequences of phonemes that can come from anywhere in the source repository, we would like the training sequences to cover as much of the repository as possible. Therefore, we start from the first phoneme \( s_1 \) in the repository, and find the longest sequence \( S = (s_1, \ldots, s_k) \) for which there is at least one corresponding sequence \( T = (t_1, \ldots, t_K) \) in the target video where \( t_i \) and \( s_j \) belongs to the same viseme group (i.e. phonemes that require the same lip expressions are in the same viseme group). We take up to two best matches in the target video with respect to \( C_{\text{match}} \) score defined in Section 3.3, and call them \( T^A = (t^{A}_1, \ldots, t^{A}_k) \) and \( T^B = (t^{B}_1, \ldots, t^{B}_k) \). We add the pairs \((S, T^A)\) and \((S, T^B)\) to the set of phoneme sequences in correspondence, and continue the scan through the source repository at \( s_1 \), until we finish scanning through the entire repository for subsequence matches (Figure 4). To quickly find the best \( T^A \) and \( T^B \) sequences in the target video, we apply fast search techniques discussed in Section 3.3. Finally, to convert each resulting phoneme sequence

ACM Trans. Graph., Vol. 1, No. 1, Article 1. Publication date: January 2021.
pair into a parameter sequence pair, we linearly retime the target video phoneme interval to the corresponding repository interval and similarly interpolate the expression parameters. The retargeting model is trained once for each new target video.

**Neural Network Architecture.** We employ a recurrent neural network (RNN) manually unrolled for $T$ time-steps to encode the temporal dynamics of the facial expressions and regress from source parameters to target parameters (Figure 5). The resulting network takes as input $T$ frames of 64 expression parameters for the source actor, and outputs $T$ frames of 64 parameters for the target actor. At the core of the network is the recurrent unit that is made up of 3 fully connected layers of 1024 nodes with relu activation. The inputs to the recurrent unit are 64 expression parameters of the current time-step, as well as the parameters and outputs of the recurrent unit for the previous $H$ time-steps. To facilitate learning of deviations from the identity transformation, the output layer of the recurrent unit with 64 nodes and no activation produces the residual values that are added to the input source parameters element-wise to obtain the prediction for the target parameters for the current time-step. We zero-pad the unavailable inputs at the first $H$ time-steps. We empirically found that setting $H = 2$ and $T = 7$ produced high-quality retargeting results.

**Loss function.** Our loss function is a linear combination of a data term and a temporal regularizer with regularization weight $\lambda$:

$$\mathcal{L} = \frac{1}{T} \sum_{i=1}^{T} \| F(s_i) - t_i \|_1 + \lambda \| F^{(2)}(s_i) \|_2$$

where $s_i, t_i$ are 64-dimensional vectors representing the $i$th time-step of the source and target parameters respectively, $F(s_i)$ is a 64-dimensional vector of the predicted target actor parameters and $F^{(2)}(s_i)$ is a 64-dimensional vector that is the second temporal difference vector, or acceleration, of the predicted values. Empirically we found that using an $L_1$ norm for the data term significantly outperforms $L_2$ by generating more expressive motions and better preserving mouth closures. The temporal regularization term is needed to make the network predict temporally stable parameters.

**Hyperparameters and training.** Since the network takes a fixed-size input of $T$ frames, we run a sliding window on each matching parameter sequences to obtain the training examples. Experimentally we set the temporal window $T = 7$ frames. For training we set $\lambda = 10$, and dropout rate at 25%, 50% and 25% for the three layers in the recurrent unit, respectively. To train the network we use stochastic gradient descent with the Adam solver [Kingma and Ba 2015] and set an initial learning rate of 0.0002 with an exponential decay rate of 0.5. We employ gradient clipping [Pascanu et al. 2013] to avoid exploding gradients. We train the network with minibatch size 100 and training typically converges within 100 epochs.

**Inference.** At inference time, we convert a sequence of source actor expression parameters into target actor parameters. Since our retargeting model accepts fixed-size $T$ frames of input and produces $T$ frames of output, we run a sliding window of length $T$ over the new sequence of source actor expression parameters at inference time. Each frame is covered by exactly $T$ such sliding windows. In order to obtain a more temporally stable output, at each frame we average the $T$ outputs produced by those $T$ sliding windows as the final output of the frame. The result is a synthesized sequence of target actor expression parameters that animate the face to speak the new content of the edit with the desired timing. We then proceed to expand these expression parameters into parameters for the whole head and use neural rendering to generate the video frames (Section 3.2).

Training our retargeting model typically requires 2–3 minutes of target actor video speaking arbitrary speech to produce high-quality synthesis results. Retargeting allows our tool to leverage the large repository of controlled source actor video (speech consists of TIMIT sentences) to generate the target actor lip motions and opens our tool to many practical applications where large amounts of controlled target actor video are not available.

### 3.5 User Controls

This speed of our synthesis pipeline opens the door to interactive user controls for iteratively refining the edit and further manipulating the facial performance.

**Refinement Control: Smoothing jumpy transitions.** Our synthesis pipeline stitches together different subsequences of expression parameters from the source repository by smoothing over a window of 4 frames around the transition boundary (Section 3.3 Fast Phoneme Search and Stitching). At times however, some transitions may still appear jumpy even after this smoothing. We allow the user to inspect the result and further refine it by manually specifying the interpolation radius (in number of frames) at user-specified transition boundaries to better smoothly out visibly jumpy transitions.

**Refinement Control: Adjusting mouth closure.** As noted in Section 3.3 mouth closure on ‘m’, ‘b’ and ‘p’ phonemes is crucial for the mouth motions to appear realistic. Thus our stitching procedure automatically inserts 2 closed mouth frames at the beginnings of these three phonemes to ensure the mouth closes correctly for them. We further allow users refine any synthesized result by extending (or reducing) the length of the inserted closed mouth frames.

**Performance Control: Inserting mouth gestures.** Users can also insert non-verbal mouth gestures (e.g. a smile) into an edit. To enable such performance control we manually annotate mouth gestures including rest, closed-mouth smile, regular teeth-showing smile, big
open-mouth smile, sad, scream, mouth gesturing left and mouth gesturing right, in the repository video. These segments can then be retrieved by our fast phoneme search just like any other phoneme annotation.

Since the annotations are on the repository, this manual annotation only needs to be done once during repository preprocessing. Note however, that users do not label the target video with these mouth gestures and our retargeting network is never explicitly trained with corresponding pairs of mouth gesture frames between the repository and target videos. Nevertheless, we have found that our retargeting network is able to generalize to unseen expressions and produce good quality expression parameters for the target actor.

With these annotations, the user can add special mouth gesture directives like [smile] anywhere in their edit of the transcript, and our tool constructs a “generalized phoneme” edit sequence \( W \) that contains phonemes and such directives. Any mouth gesture that appears in \( W \) is given a default duration of 0.5 seconds that the user can override with an explicit duration e.g. [smile:1.5s]. We employ a special substitution cost in \( C_{\text{match}} \), described in Section 3.3 for “gesture phonemes” that is set to infinity for a non-match to ensure that we retrieve the correct “phoneme” match for the gesture. When there are multiple candidates, \( C_{\text{match}} \) takes duration into account and picks the gesture with duration closest to the query. The rest of the editing pipeline (Section 3.2) is otherwise unmodified.

**Performance Control: Adjusting speaking style.** Our tool allows the user to select a different speaking style for the synthesized result by using a version of the repository with the desired style. In addition to the default repository which captures a “neutral” speaking style, we have recorded an “energetic” repository of our source actor with significantly less mouth movements. Figure 6 (third row) shows frames from these alternative repositories.

Importantly, we do not have to retrain our neural retargeting model (Section 3.4) for each additional style repository. We train this retargeting model once using only the default neutral repository. We have found that our default retargeting model can extrapolate to retarget subsequences of source actor face parameters retrieved from other speaking style repositories of the same actor. Moreover, the other repository videos can be captured at different times, with different background and the source actor can even be wearing different clothes or have a different hairstyle. Thus, our tool generates videos with different speaking styles by using one of the alternative style repositories in the fast phoneme search step, but leaves the remainder of the synthesis pipeline unchanged.

### 3.6 Implementation Details

We implemented the fast phoneme search in Python and both our neural retargeting model and the GAN renderer in Tensor-Flow [Abadi et al. 2015]. The monocular head tracker and renderer are written in C++ with shader language extensions.

In preprocessing the repository and target videos, phoneme alignment takes one third of the video time, and face registration takes 110 ms per frame. It takes 30 minutes to generate training data for our neural retargeting model and another 30 minutes to train it on one NVIDIA GTX 1080Ti. Training the GAN for neural rendering takes 17 hours on one NVIDIA Tesla V100.

In our synthesis pipeline, our fast phoneme search requires 5 seconds for a typical edit of 5 words containing 20 phonemes. Retargeting inference speed is 10K fps. Composite images are rendered at 12 fps and final GAN rendering takes 7 fps on two NVIDIA GTX 1080Ti. All together, a typical 5 word edit takes around 30 seconds for the full video generation (Table 1).

It should be possible to further reduce the feedback time by parallelizing our synthesis pipeline. Phoneme search could be distributed where each worker job is responsible for searching a fraction of the repository. Both parts of the neural rendering step – forming the composite images from target actor head parameters and applying the GAN to generate photorealistic frames – are parallelizable by distributing the frames. We have performed initial experiments on parallelizing the GAN rendering which is the main bottleneck in our pipeline. Distributing the GAN rendering across a cluster of 8 NVIDIA Tesla V100s achieved a rendering rate of 24fps, a 3.4x speedup from the original 7fps for this step. Note that this speed up rate includes image compression overhead. Overall this experiment cuts the end-to-end synthesis time from 40 to 20 seconds for a typical 8-word sentence. Similarly parallelizing the other parts of the pipeline and using sufficient hardware we believe that the end-to-end video generation feedback time could be reduced significantly. Streaming the frames as they are ready could also further reduce the latency from issuing an edit to seeing the first frames of the result, enabling real-time interactive editing sessions.

### 4 RESULTS

Figures 1 and 6 show examples of iterative text-based editing sessions for a variety of target videos including recordings of graduate students, YouTube video and a take from filming a dialog scene. We encourage readers to watch the videos in our supplementary materials to see how our text-based interface facilitate the iterative editing workflow used in each of these sessions.

**Session 1: Talking-head with glasses.** Our first session works on a 2.5 minute target video (Figure 1). The editor explores ways for the actor to parody Martin Scorsese and express that Marvel movies are not to be considered real cinema. They first synthesize “Marvel movies are not cinema” from a resting pose. Feeling it is too blunt, they slightly change the wording to “Marvel movies are not really cinema”, and eventually settle on the firmer statement “Marvel movies really are not cinema”. They then insert a smile at the end to soften the overall tone. Our tool is able to produce results with synchronized mouth motions at each step.

| #words | #phonemes | #frames | search (s) | render (s) | total (s) |
|--------|-----------|---------|------------|------------|----------|
| 1      | 4         | 24      | 1.51       | 9.96       | 12.39    |
| 3      | 15        | 49      | 2.67       | 12.94      | 20.30    |
| 6      | 25        | 72      | 5.32       | 17.74      | 23.06    |
| 8      | 39        | 105     | 7.57       | 23.33      | 30.90    |
| 10     | 49        | 134     | 11.60      | 31.19      | 42.79    |

Table 1. Runtime of our tool on a variety of edit lengths. Search time scales roughly linearly with the number of phonemes, and render time scales linearly with the number of frames. Even for long edits of 10 words, our system can generate video in approximately a minute.

**Session 2: Editing a superhero.** For our second session, the editor takes a Marvel movie trailer and applies a series of edits to parody the Marvel movie world. The editor starts with the firmer statement “Marvel movies are really not cinema” and then softens the tone to “Marvel movies are not really cinema” and finally “Marvel movies are not cinema” (Figure 6). In this session, we encode and decode the speech to the target actor’s pronunciation, and show some constraints on the talking-head visual appearance.
Session 2: Talking-head with stubble. Our second session works on a 3.5 minute target video (Figure 6 first row). The editor explores ways for the actor to give the instruction to start the time warp by jumping to the left then stepping to the right. They first synthesize the instruction phrase, then add gestures “[mouth_left]” and “[mouth_right]” to the corresponding location in the dialog. Next, feeling the gestures go too quickly, they lengthen the gestures to one second each. Our tool produces realistic video with mouth motions synchronized to the audio and the gesture directives.

Session 3: Movie scene. Our third session works on a target video of a single take from a dialogue scene (Figure 6 second row). The target video is a challenging one because it is only 1 minute long, and the actress speaks for only 30 seconds in the take. Our tool nevertheless is able to synthesize compelling results for this session. In the session, the editor prototyes ways for the actress to express her hatred towards the murderer of her sister’s dog. They first try “I will make him suffer”. Then for added creepiness, add a tight-lipped smile of 1 second duration at the end. Finally, they settle on a less hostile line instead. While the neural renderer struggles with the lack of data to produce images as sharp as those from the previous two sessions, our tool is still able to produce realistic and synchronized mouth motions that give the user a good sense of how the scene would look with the alternative line and gesture.

Session 4: Interview. Our fourth session works on a 5 minute excerpt of a YouTube video (Figure 6 third row). The editor explores different delivery styles for the phrase “may the force be with you”. They first synthesize the phrase with the default repository, then switch to a mumble style and finally to an energetic style. While the mouth movements match the audio, there is visibly less motion with the mumble style and more motion with the energetic style.

5 EVALUATION
To evaluate our tool we analyze the quality of the synthesized video as we vary the algorithmic methods (e.g. fast phoneme search, neural retargeting) used in our synthesis pipeline, and as we vary the data (e.g. length of target video, repository or edit) provided to the algorithm. We then compare the quality of our results to those of previous work. Finally we report on user studies that quantitatively evaluate the quality of our synthesized results. Unless otherwise
5.1 Varying the Algorithmic Methods

Comparison of phoneme search and stitching methods. We compare the impact of using ablated versions of our fast phoneme search and stitching algorithm (Section 3.3) with the phoneme search method of Fried et al. [2019]. More specifically, because their synthesis pipeline does not include neural retargeting we treat Fried et al.’s approach as a baseline method and build two comparison pipelines. The first one adds only phoneme context expansion (Section 3.3) to the baseline stitching method. The second pipeline replaces their phoneme search and stitching method with the full version of our fast method (we use “Modified Fried et al. [2019]” to denote this second version of the pipeline with our fast method in later comparisons). All three pipelines assume access to an hour of target video which serves as the repository. Figure 7 and videos in the supplemental materials show that results generated using our fast phoneme search and stitching method are often indistinguishable from the those generated by the baseline. The main differences that do appear are often subtle as our forced mouth closure on \( m, b, \) and \( p \) phonemes reduces open mouth artifacts and our new context aware stitching (Section 3.3) across subsequence transitions yields smoother, less jittery lip motions. We further compare our method against Modified Fried et al. [2019] with user studies in Section 5.4. More importantly our method takes only seconds to run, which is three orders of magnitude faster than the hours required by baseline Fried et al. [2019], making it possible for the user to iterate on edits.

Comparison of retargeting methods. Our neural retargeting method (Section 3.4) transforms a sequence of source actor expression parameters to those of the target actor. We compare our method with two simpler baseline retargeting methods. The copy baseline directly copies the source actor parameters to the target actor. The linear baseline replaces our retargeting network with a linear model. More specifically, we manually chose 2.5 minutes of target video containing phrases that exactly matched phrases in the source repository and established a frame-to-frame correspondence by re-timing the target phonemes to match the lengths of the source phonemes. We then applied linear regression on the source and target parameter pairs to obtain the linear baseline model. Figure 8 and the supplemental materials show that our neural retargeting model produces the best results, while direct copying produces uncanny mouth shapes and the linear model often fail to close the mouth, causing out-of-sync lip motions.

5.2 Varying the Amount of Data

Varying the length of the target video. We examine the effect of different amounts of target video by applying our tool on 10 minute, 5 minute, 2.5 minute, 1 minute and 30 second subsets of a target video. More target video generally results in sharper images, higher
quality mouth interiors and smoother mouth motions. But the difference is subtle when target video exceeds 5 minutes, and at 2.5 minutes the results remain plausible. With a 30 second target video however, although the results still have well-synchronized mouth motions, our neural renderer struggles and produces noticeably blurrier images, as shown in Figure 9 and videos in supplemental materials. Please zoom in on the front teeth to see the difference.

5.3 Comparisons to Other Methods

Comparisons with Synthesizing Obama [Suwajanakorn et al. 2017]. Suwajanakorn et al. [2017] have presented a technique for taking audio speech of Obama as input and synthesizing a video of Obama saying the speech. We compare our results to theirs using only 3 minutes of video of Obama. As shown in Figure 11, our approach gives more plausible and synchronized mouth motions compared to
Comparisons with Neural Voice Puppetry [Thies et al. 2020]. Concurrent to our work, Neural Voice Puppetry (NVP) can synthesize talking head videos from audio signal input, or from text by using a synthetic voice to obtain the audio signal. Given an audio speech track, we compare our result to theirs by applying our method on the phoneme timings extracted from the audio. Figure 13 and videos in supplemental materials show that while both approaches generate mouth motions that synchronize with the audio, our fully automatic result (without user refinement) generates closed-mouth frames at the desired phonemes (\m, \b, \p), whereas Neural Voice Puppetry leaves the mouth open for many of these phonemes. In addition, unlike NVP, our tool allows the user to iteratively refine the automatic results and adjust the performance. We further compare our results to NVP in the user studies in Section 5.4.

Comparisons with Chen et al. [2019] and Vougioukas et al. [2019]. Both Chen et al. [2019] and Vougioukas et al. [2019] generate talking head videos from audio input and a single frame of the target actor. Given an audio speech track, we compare our result to theirs by applying our method on the phoneme timings extracted from the audio. Figure 14 and videos in supplemental materials show that while all three approaches produce good lip-sync with proper mouth closures, both Chen et al. [2019] and Vougioukas et al. [2019] produce videos of less resolution than our result. In addition, their results do not have the natural head motion in our result, and by always centering the video around the cropped head, their results can contain warping artifacts in the background, making them ill-suited for incorporation into a video-editing workflow.

5.4 User Studies and Automatic Metrics

We use both user studies and automatic metrics to quantitatively evaluate the quality of the video generated by our editing tool. In the user studies, we investigate both short and long edits, while ablating the target video length and the neural retargeting step. We compare to previous work [Fried et al. 2019; Thies et al. 2020] and to ground-truth video. We follow the study design used in previous talking-head synthesis research [Fried et al. 2019; Kim et al. 2018]. Specifically, participants see one video at a time in randomized order and are asked to rate the statement “This video clip looks real to me” on a 5-point Likert scale ranging from strongly disagree (1) to strongly agree (5). All videos used in our studies are available in the supplemental material.

User study 1: Short Phrases (1 – 4 words). Short phrases are the main type of result shown in Fried et al. [2019]. Such edits are useful for minor fix-ups on existing sentences. In this study we compare our automatic synthesis results (“Ours”) to 3 versions of Fried et al. [2019]. (1) Their method with the same amount of target video as used by our tool, which is less than 5 minutes in all cases. (2) Their method with 1 hour of target video, which is their recommended amount, and more than 12 times the amount we use. (3) A version of their method with our fast phoneme search and stitching algorithm (Section 3.3), but with 1 hour of target video (“Modified Fried et al. [2019]” in Section 5.1), to evaluate the effect of ablating our neural retargeting step. We also compare to ground truth video recordings. We recruited 110 participants to view 25 videos each (5 conditions
for each of 5 edits). We report Likert scale responses in Table 2 ("Short Phrase"). The differences between all pairs, except "Ours" vs. "Modified Fried", are statistically significant. All p-values have been adjusted for multiple testing and are reported in supplemental materials.

Our tool outperforms Fried et al. [2019] both when using 5 minutes of data and 1 hour of data. We believe this is due to our results having more accurate mouth closures and better temporal coherency in mouth motions. Results are similar for our tool and Modified Fried, indicating that our neural retargeting step does not have much negative effect on result quality. Together these results also suggest that our fast phoneme search with stitching that forces closed mouths for \(\text{m}, \text{b}, \text{p}\) phonemes leads to higher-quality synthesis than the slow phoneme search and stitching approach used originally by Fried et al. [2019]. Although a gap still remains between our synthesized results and ground-truth videos, our results for short edits are rated as real almost two thirds of the time.

User study 2: Full Sentences (6 – 9 words). Full sentence synthesis is more challenging, since longer synthesis equates to a larger chance of inaccurate matches and synthesis artifacts. However, synthesizing full sentences as opposed to short phrases opens up more use cases (Section 4). Investigating full-sentence synthesis also emphasizes the quality differences between methods. The conditions in this user study are the same as for user study 1. We recruited 153 participants to view 25 videos each (5 conditions for each of 5 sentences). We report Likert scale responses in Table 2 ("Full Sentence"). The differences between all pairs, except "Fried < 5 min" vs. "Fried > 1 hr", are statistically significant. All p-values have been adjusted for multiple testing and are reported in supplemental materials.

Our tool produces the highest-quality results, followed by Modified Fried with over 1 hour of data, then by Fried et al. [2019]. Similar to user study 1, here our results have better mouth closures and smoother mouth motions than Fried et al. [2019]. It is worth noting that our results are even better than Modified Fried. We believe this is because our tool has a higher-quality source repository which becomes more salient when the edits are long. It shows the advantage of our approach to decouple source repository from target video, as data quality improvements to the repository can benefit many different target videos. The one-time cost of building a high-quality repository amortizes across all the edits that use it.

User study 3: our tool vs Neural Voice Puppetry [Thies et al. 2020]. The third user study compares our results to those of Neural Voice Puppetry [Thies et al. 2020], where we show viewers videos generated by the two methods from the same audio speech track. We recruited 90 participants to view 8 videos each (4 from each of the two methods, Table 3). The audio tracks used in the videos are not the actor’s real voice, and we believe this is the predominant reason for overall lower scores (for both methods). The difference between conditions is not statistically significant, and our results have similar mean scores to those of NVP. Nevertheless, as mentioned in Section 5.3, closely examining the videos generated by the two approaches, we find that our method often does a better job of closing the mouth on \(\text{m}, \text{b}, \text{p}\) phonemes. We also note that while our user studies evaluate our automatic results, unlike NVP, our tool also provides refinement and performance controls that can be used to improve results over the course of an interactive editing session.

**Automatic Metrics.** For videos in user study 1 and 2 where we have ground-truth recordings, we further evaluate the results using automatic metrics against growth truth videos. To measure reconstruction quality, we compute structural similarity index (SSIM [Zhou Wang et al. 2004]) and peak signal-to-noise ratio (PSNR). To measure lip motion quality, we compute the Landmarks Distances (LMD) [Chen et al. 2018]. We report the results in Table 4. Although our methods achieve favorable scores on many of these metrics, the score differences are quite small and visual differences can be subtle. We believe the user studies provide a better and more trustworthy measure of video quality.
We have demonstrated an iterative text-based tool for editing talking-head editing scenarios in which only a few minutes of target actor video could reduce the feedback loop time significantly. However, our approach does have several limitations that could be addressed in future work.

**Further reduce feedback loop time.** Our tool currently requires about 30 seconds to synthesize a typical 5 word edit. While this feedback loop time allows users to try a variety of edits and refinements, seeing a synthesized result immediately (in real-time) would allow even more iteration and exploration of design space. As noted in Section 3.6, parallelization of our fast phoneme search and neural rendering steps as well as streaming playback of the synthesized video could reduce the feedback loop time significantly.

**Improve quality of synthesis results.** Although our method compares favorably in quality with previous talking-head synthesis techniques (Section 5.4), there is still a gap in realism between our results and ground-truth videos. As our method relies on a rich repository of source video to provide mouth motions for phoneme coarticulations, it may be possible to improve synthesis by developing higher-quality repositories. One approach may be to leverage existing work in text-driven 3D human mouth animation [Edwards et al. 2016] to render unlimited amounts of mouth motions to serve as the repository. Another direction is to build multiple repositories of many different source actors and then given a target video, develop techniques to pick the best source actor for the target.

**Performance controls over full face.** Our current approach focuses on synthesizing lip motions that match the target edit. While our tool offers controls for inserting mouth gestures and changing the speaking style, the effects of these controls are limited to the lower part of the face. Others have demonstrated techniques for controlling more of the head, including the ability to change head pose, gaze direction and whole face expressions [Kim et al. 2018] However, these techniques often introduce artifacts in the hair and with the clothes at the neckline. Adding such full face controls in an artifact-free manner remains an open research direction.

Table 4. Results of automatic metrics. We compare our results to 3 versions of Fried et al. [2019] for both short phrase edits and full sentence syntheses. Best score is bolded. Our tool tops SSIM and PSNR for full sentence syntheses and ranks second after Modified Fried for LMD on full sentence and PSNR on short edits.

| Condition          | Length | SSIM     | PSNR     | LMD      |
|--------------------|--------|----------|----------|----------|
| Fried et al. [2019]| < 5 min| 0.89929  | 25.08146 | 4.57492  |
| Fried et al. [2019]| > 1 hr | 0.89925  | 25.09057 | 4.47108  |
| Modified Fried     | > 1 hr | 0.89934  | 25.09670 | 4.38139  |
| Ours               | < 5 min| 0.89921  | 25.09349 | 4.57060  |
| Fried et al. [2019]| < 5 min| 0.96339  | 32.74630 | 3.75896  |
| Fried et al. [2019]| > 1 hr | 0.96552  | 32.94852 | 3.67367  |
| Modified Fried     | > 1 hr | 0.97578  | 35.00073 | 2.90959  |
| Ours               | < 5 min| 0.97630  | 35.12082 | 3.18670  |

Figure 3. Median lengths of generated generations.

6 LIMITATIONS AND FUTURE WORK

We have demonstrated an iterative text-based tool for editing talking-head dialogue and performance that can be applied to many real-world editing scenarios in which only a few minutes of target actor video is available. However, our approach does have several limitations that could be addressed in future work.

**Further reduce feedback loop time.** Our tool currently requires about 30 seconds to synthesize a typical 5 word edit. While this feedback loop time allows users to try a variety of edits and refinements, seeing a synthesized result immediately (in real-time) would allow even more iteration and exploration of design space. As noted in Section 3.6, parallelization of our fast phoneme search and neural rendering steps as well as streaming playback of the synthesized video could reduce the feedback loop time significantly.

**Improve quality of synthesis results.** Although our method compares favorably in quality with previous talking-head synthesis techniques (Section 5.4), there is still a gap in realism between our results and ground-truth videos. As our method relies on a rich repository of source video to provide mouth motions for phoneme coarticulations, it may be possible to improve synthesis by developing higher-quality repositories. One approach may be to leverage existing work in text-driven 3D human mouth animation [Edwards et al. 2016] to render unlimited amounts of mouth motions to serve as the repository. Another direction is to build multiple repositories of many different source actors and then given a target video, develop techniques to pick the best source actor for the target.

**Performance controls over full face.** Our current approach focuses on synthesizing lip motions that match the target edit. While our tool offers controls for inserting mouth gestures and changing the speaking style, the effects of these controls are limited to the lower part of the face. Others have demonstrated techniques for controlling more of the head, including the ability to change head pose, gaze direction and whole face expressions [Kim et al. 2018] However, these techniques often introduce artifacts in the hair and with the clothes at the neckline. Adding such full face controls in an artifact-free manner remains an open research direction.

7 ETHICAL CONSIDERATIONS

Our editing tool is designed to enable an iterative workflow for removing filler words, adjusting phrasing, or correcting mistakes in a talking-head video. While such tools can facilitate content creation and storytelling, tools like ours, that let users manipulate what a target actor is saying, can also be misused. We follow the guidelines suggested by Fried et al. [2019] for ethically using such tools. (1) Video generated by our tool should be transparent about the fact that it has been manipulated. (2) Actors must give consent to any manipulation before a resulting video is shared widely.

We also recognize that these guidelines alone will not stop bad actors from using tools like ours to create false statements and slander others. Therefore, it is also critical for researchers to continue developing tools for detecting, fingerprinting, and verifying such video manipulation. Openly publishing the technical details of our tool can increase public awareness and help detection efforts. Ultimately these issues may also require regulations and laws that penalize misuse while allowing creative and consensual use cases.

8 CONCLUSION

Iterative editing is central to many content-creation tasks and is especially common in video editing. We have shown how to enable such iterative editing in the context of editing talking-head video using a text-based interface that allows changes to wording and facial performance while providing refinement controls. Whether an editor trying different ways to phrase the dialogue in a film, developing dialogue for a conversational agent, or correcting a mistake in a lecture, such iteration is often essential for finding the most appropriate result. We believe such tools that facilitate video editing can democratize content-creation and enable many more people to tell their stories.

9 ACKNOWLEDGEMENTS

This work was supported by the Brown Institute for Media Innovation and National Science Foundation (1908727).

REFERENCES

2020a. Google Cloud Speech to Text API. https://cloud.google.com/speech-to-text
2020b. Google Cloud Text to Speech API. https://cloud.google.com/text-to-speech
2020. Lyrebird AI. https://www.descript.com/lyrebird-ai
2020. Rev. https://rev.com
2020a. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/Software available from tensorflow.org

ACM Trans. Graph., Vol. 1, No. 1, Article 1. Publication date: January 2021.
ACM Trans. Graph., Vol. 1, No. 1, Article 1. Publication date: January 2021.