Abstract

We provide the following materials in the appendix:

- *A full broader impact statement (Section A)*
- *Details about our model architecture (Section B)*
- *Details about how we provide video data into the model, including how we align the modalities and perform the masking (Section C*
- *Details about how we adapted our model to downstream tasks (Section D)*
- *Details about how we collected data (Section E)*
- *Additional experiments (Section F)*

A. Broader Impact Statement

In this paper, we have presented a model for learning multimodal neural script knowledge, through incorporation of audio as a first-class citizen alongside text and video frames. We argue that academic study of this learning paradigm is important, in part because it relates to how we as humans understand the world. We as humans process situations by perceiving through multiple modalities and interpreting the result holistically.

At the same time, the work and methodology that we outlined risks dual use. Like other large machine learning systems pretrained on web data, our system may reproduce harmful social biases present in its training data. While a variety of past work has studied risks of *language-only* pretraining [127, 14, 7, 61], the video-centric pretraining that we explore in our work might have different benefits and risks. We discuss these below, along with how we worked to mitigate them through our work.

A.1. Privacy.

A significant risk with training on data at YouTube scale is protecting user privacy. We took several proactive steps to ensure this, that in turn build off prior work and community norms [1, 84, 128]:

a. We release only the video IDs for download, following prior work [1, 84]. Thus, if a user deletes a video off of YouTube, it becomes removed from YT-Temporal-1B as well, giving content creators a right to opt out of all uses of their videos.

b. Building off of past work [128], we directed our data collection towards *public* and *monetized channels*. These channels are identifiable insofar as they contain more subscribers, and more videos. They include companies that have official accounts, including journalism outlets like the *New York Times* and *Vox*. They also include individuals for whom making public YouTube videos is their full time job. In either case, our use videos in question for research purposes can be seen as *fair use*.

Model	Accuracy (%) Voice	Image+Voice	Image
RESERVE-L	10.8	9.6	10.7
CLIP ViT-B/16 [92]			**86.0**

Table 6: Zero-shot person (face/voice) recognition accuracy on VoxCeleb2 [87] and VGGFace2 [17], using different modalities. While RESERVE can perform person recognition from several modalities, its performance is much lower than the recognition-optimized CLIP model in the image-to-name setting. We hypothesize that this is due to a similarity between this setting and CLIP's pretraining data – news articles often include celebrity images, paired with their names.

Framing of privacy. Privacy is a nuanced topic with many societally, culturally, and generationally-specific interpretations. We took inspiration from Marwick and Boyd [83]'s framework of *networked privacy*, which posits that users posting public videos might *encode* private information – enough so that their intended viewership (friends, possibly) can catch the gist, but not enough so as to leak private details like phone numbers to the world.

Through the lens of networked privacy, we see key differences between studying videos on a moderated platform, versus NLP work that trains models from the open web (e.g. [29, 93, 16]). When YouTube users upload videos, they tend to understand details of its privacy policy, beyond consenting to it [65]. Likewise, YouTubers typically upload their own videos [102]; the platform deters users from re-posting other users' content. These factors differ from text on the open web. Today, 'data brokers' post private details (like phone numbers) to the web for profit [25]; concerningly, a study on language models suggests that models are vulnerable at *memorizing* this private information [18].

It is worth examining our research through other framings of privacy as well. For example, internet platforms profit off of user data, whereas users do not share equally in these profits [37]. For this, and for the other reasons mentioned, we aim to release our model only for research-based use.

A.1.1 Empirical study: can RESERVE identify individual celebrities?

Inspired by work studying language model memorization of private information [18], we wish to empirically probe RESERVE's ability to recognize individuals. Our goal during model development was **not** to optimize for this ability. Instead, our goal was to study models for *multimodal script knowledge* (what people might be doing in a situation over time, and why) instead of long-tailed *visual recognition* (including who those individuals are). These goals might

trade off – for instance, our training data only has individuals' names when they are mentioned in ASR subtitles, a pairing that might be significantly noisier than images and text on the open web.

We study this capacity on the VoxCeleb2 and VGGFace2 datasets [87, 17], where we created a test set of 120 celebrities, with 100 samples of each. We study these datasets not to promote them, but to establish a **conservative upper-bound** for the capacity of a model to recognize non-celebrities. We hypothesize that if RESERVE struggles to select the right celebrity out of 120 predefined options, it would struggle much more at identifying random people (where the set of candidate names is much greater). We test this hypothesis over three zero-shot settings:

1. **Voice to name.** Given an audio clip sampled for a celebrity, we encode it with our model's audio encoder. We provide our model's joint encoder the text '`the sound of MASK`', followed by the encoded audio. A blank image is provided. We extract the representation on top of the `MASK`, and choose the most similar celebrity name.
2. **Image+voice to name.** Here, we adopt the same format as 'Audio to name,' except we additionally encode an image of the celebrity's face in question.
3. **Image to name.** Here, RESERVE encodes an image of the celebrity in question, and we provide it with text 'A picture of `MASK`.' No audio is provided. Using our model's joint encoder, we select the closest encoded celebrity name, out of all options.

   We use this format to compare to a CLIP model, which was trained on web images with captions [92]. For the CLIP comparison, we use it to encode each image, and for all considered celebrity names, the sentence 'A picture of `${name}`'. We choose the closest encoded sentence to the encoded image.

We show our results in Table 6. In all modes, our model is less than 11% accurate at recognizing celebrities. Curiously, the accuracy *drops* given both the image and the voice, suggesting that the way we fused a celebrity's image and voice together might be outside the model's training distribution. These results are significantly lower than CLIP's 86% accuracy at classifying a person from their image.

In Figure 6, we investigate more into which celebrities our model is best at recognizing. Only a few celebrities are reliably classified; these tend to be very famous celebrities like Oprah Winfrey and Justin Bieber. Several sports players are recognized well (including Lebron James and Roger Federer), which could imply that our model learned their identities from watching sports replays or commentary. Most other celebrities are hardly recognized, whereas CLIP does well across the board.

Results summary. Together, these results show that while models like CLIP focus on encyclopedic knowledge

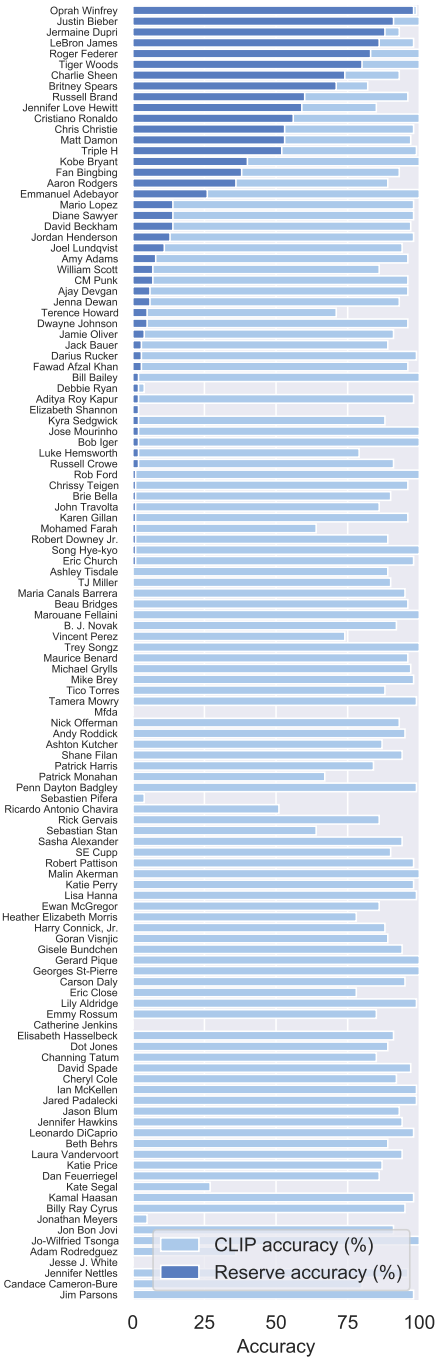

Figure 6: VoxCeleb2 results per-celebrity, comparing RESERVE-L versus CLIP ViT-B/32 in the same 'image-text' setting. Our model reliably recognizes A-list celebrities like Oprah Winfrey, very famous musicians (Justin Bieber) and sports players (LeBron James). However, it struggles on every other celebrity, particularly compared with CLIP. This suggests that our model primarily learns **semantic** as opposed to **recognition-level** encyclopedic knowledge.

that results in strong zero-shot person recongition accuracy, RESERVE is not as effective as other models in memorizing particular celebrities– and, thus, perhaps not as effective as memorizing particular *non-celebrities*. These results suggest that RESERVE's objectives and data might make it *less of a concern* to release privacy-wise, versus models trained on web images with captions.

As the rest of the paper emphasizes however, RESERVE performs well on tasks with temporal understanding and commonsense reasoning as the primary goal. On a broader level, these results suggest that it is possible to learn strong models about temporal reasoning *without* person-level memorization, though more work is needed.

A.2. Biases in (pre)training data.

The 'knowledge' that our model learns should be viewed as situated within YouTube [67], which has numerous biases (that we will discuss next). Past work has made similar observations for language model pretraining on the open web[7]. One of the root causes of such bias is learning objectives that encourage memorization of surface level cooccurences, rather than truly causal factors [56, 8, 117]. Though it is possible that in the very long term, a paradigm of *grounded learning* might resolve some of these issues, the objectives in this work still likely reify biases that exist in the YouTube data.

Platform biases. Unlike many other pretraining efforts, that scrape data from the open internet (e.g. [93, 16, 92]) which directly leads to toxic biases (e.g. [42, 32, 11]); we trained our model on YouTube, which is a moderated platform [101]. Though the content moderation might perhaps reduce overtly 'toxic' content, social media platforms like YouTube still contain harmful microagressions [15], and alt-lite to alt-right content [95]. Additionally, it should be mentioned that the content moderation on YouTube disproportionately filters out minoritized voices [44]. Thus, despite us not using any word-based 'blocklist,' our model's pretraining data is still biased [32]. Even without videos being explicitly removed, the 'YouTube algorithm' incentivizes the production of certain types of content over others [12, 102]; e.g. people's roles in YouTube videos tend to be highly gendered [86], which might bias situation understanding [133].

Bias amplification. In this work, we pretrained a model primarily on ASR text, which is itself produced by another model. The automatic captions in YouTube are known to suffer from gender bias [107], which our model (like neural models generally) might in turn *amplify* [133]. The transcriptions on YouTube are also likely poor at handling important identity markers, like pronouns. Already, text-only models like BERT struggle with pronouns like they/them and zi/zir; our reliance on ASR text makes our corpus likely worse in this regard [28]. While past work, namely MERLOT [128], 'cleaned' this ASR text – through another large language model – we opted not to do so for this work due to computational expense. Though in that work, the ASR-denoisification was found to boost performance in VCR, it seems unlikely that it would solve this core issue of model bias.

A.3. Dual use.

Learning connections between video, audio, and text – though an important area of study as we have argued – can be used for undesirable applications, beyond what we have outlined under 'biases.' We outline and discuss a few below.

Generating fake content. A concern for pretrained models is that they can generate fake content, that could be used by 'bad' actors for their ends [127]. It should be noted that our model cannot explicitly 'generate' text, audio, or vision in a direct sense. Nonetheless, however, it is possible that a finetuned or expanded version of this model could be used for that purpose – and that our model would be *more helpful* to such an actor versus them training their own (perhaps larger) model from scratch.

Surveillance. Our model might contain representations that enable it to be used in surveillance applications. As we note in Appendix A.1.1, our model's low performance on person recognition suggests that it might perform poorly recognition-focused applications. Still, one possibility is that a *neural script knowledge* could 'summarize' surveillance videos in some form (like identifying an activity of interest), without identifying the person(s).

We suspect (but cannot definitively prove) that the *reporting bias* of the YouTube data that it was trained on might make it poor for such a surveillance-focused task [50]. Namely, most surveillance videos are sparse in nature – finding an activity of interest is like finding a 'needle in a haystack' [91]. Though, some surveillance videos are inevitably posted on YouTube and then captioned, these disproportionately contain *interesting events* (like somebody's car crashing into a house). It is not clear whether our system could be easily adapted to such a sparse problem; the amount of work required suggests that it might be out-of-scope at least for low-skill actors. On the other hand, this broad research agenda, and perhaps all of computer vision for that matter, might enable large actors to do just that [134]; which might not be addressable through purely technical solutions [52].

Harmful outcomes if deployed. Beyond the *biases* that our system possesses, some applications of our system – if deployed in production – could cause harm, particularly to groups already harmed by AI systems. Of note, linking someone's voice with their appearance is not always a good thing [94]. Likely some of the key features that our model learns – though we did not teach it this explicitly – involve recognizing gender, and this is harmful especially to transgender individuals [55].

A.4. Energy consumption.

Our model cost a lot amount of energy to pretrain [103]; roughly 3 weeks of time on a TPU v3-512. The total carbon footprint of our work was a net 8.23 tons of CO_2 equivalent, which is roughly 4.5% of the emissions of a jet plane flying round-trip from San Francisco to New York.[9]

At the same time, it is possible that our model could save energy overall, when shared with researchers who build off of our system. Indeed, RESERVE-B uses less energy than MERLOT [128] (due to a smaller vision backbone, and smaller image sizes), MERLOT in turn is more efficient than past work which used expensive detector-based backbones (e.g. [106, 21, 131]), that are made more expensive because some of their computational primitives (like non-maximum suppression) are difficult to make efficient on-device.

A.5. Synthesis.

With these risks in mind, we release our video IDs, as well as RESERVE's checkpoints, exclusively for research use. We believe that at this point in time, we as a field lack full knowledge of the privacy, bias, and dual-use risks of video-based models – though, we hope that our analysis in this section provides a good starting point. For instance, while the objectives that we have studied were designed to promote learning general neural script knowledge above *encyclopedic memorization*, they have not yet been tested in all possible cases. By opening our models to the research community, we hope to promote fundamental work in uncovering both promising aspects of these systems, alongside examining their risks. We hope to contribute to these lines of research as well.

B. Model implementation details

In this section, we discuss at a more in-depth, technical level, how we implement certain aspects of RESERVE, and other details (like its runtime in FLOPs). We discuss our use of rotary position encodings (B.1), how we set the sequence lengths for the model (B.2), measure the model's computational footprint (B.3), list hyperparameters (B.4), and discuss several training strategies (B.5.

B.1. Rotary position encoding

We use a rotary position encoding to model the relative location of input sequences [104, 10]. We chose this primarily because we did not want to use absolute (additive) position embeddings, which would have to be added to the inputs of each encoder, and possibly at multiple levels in the hierarchy (e.g. for the joint encoder, the video segment index t would be needed as well).

The rotary encoding uses no parameters, and instead uses a kernel trick to allow the model to recover relative distances between key and query elements in a Transformer's attention head. This can be seen as 'rotating' pairs of elements; we apply the rotation to only the first *half* of each 64-dimensional head, and the second half is kept as is.

Multidimensional coordinates. We treat each token as having a 4-dimensional position of (h, w, ℓ, t), corresponding to the h, w coordinates in the image, the position ℓ in the text-sequence, and the segment index t. If a dimension is irrelevant to a modality (like h, w for text), we set it to 0. Thus, for our various encoders, we use the following coordinate schemes:

a. Video Frame Encoder (ViT): just the h, w coordinates of the image; so $(h, w, 0, 0)$.

b. Audio Encoder: Only the 1-D position ℓ of the patch in the spectrogram: $(0, 0, \ell, 0)$.

c. Text Span Encoder: Only the 1-D position ℓ of the token in the input: $(0, 0, \ell, 0)$.

d. Joint encoder: Here, we use all coordinates. Inputs from the video frame encoder have coordinates $(h, w, 0, t)$, where t is their segment index. The text and (pooled) audio inputs are merged, and they each have coordinates $(0, 0, \ell, t)$, where ℓ here is the absolute position in the entire sequence (across segments).

As part of our implementation, we normalize the rotary coordinates. h, w are scaled to be in the range $[-1/2, 1/2]$, such that text is implicitly 'in the center' of the image. Likewise, ℓ and t are scaled to be in the range of $[0, 1]$. The positions are used to compute relative distances, by using a kernel trick to rotate coordinates in the keys and values of each d_h-sized Transformer attention head.

B.2. Sequence lengths

We briefly remark on the sequence lengths used by parts of the model.

a. Video Frame Encoder (ViT): Most YouTube videos are widescreen (16x9). We thus used a widescreen resolution for our video frame encoder. It takes in patches of size 16x16, and we used a layout of 12 patches (in height) by 20 patches (in width). This corresponds to **192x320**. Among other factors that are important are ensuring that TPUs do not execessively pad the sequence length [130]. The sequence length is 241 in this case, as there is a `CLS` token, and it gets padded to 256.

Attention pooling. As we note in the main text, after-

[9] **CO_2 Calculation.** It is also important to consider the *location* where these TPUs are located, as the renewables portion at each datacenter is not equal [88]. Our TPUs were in the 'europe-west4' region, which uses on average 60% carbon-free energy, and a Grid Carbon intensity of 0.410 kgCO_2eq / kWh. A single TPU v3 processor (with 8 cores over 2 chips) has a power average of 283 W, so after performing the math from [88], our training cost 20,000 kWh. This gives us a net 8.23 tons of CO_2 equivalent. It should be mentioned that this figure only covers the *electricity usage* given the chips (and the datacenter), not the raw materials involved in making these chips (which is significant [111]).

wards we apply attention pooling in a 2x2 grid (ignoring the CLS token here). Similar to Transformer-style query,key,value attention [110], the query is the average of the vectors in the 2x2 grid; the keys and values are learned projections of the vectors. This gives us a $H/32$ by $W/32$ grid for the joint encoder (6 x 10).

b. Audio Encoder. Our model independently encodes each 1.6 second of audio (a segment has three such 'subsegments'). We do this through spectrograms. Each window involves 1536 samples at a sample rate of 22500 Hz, and there are 588 samples 'hops' between windows. We chose these hyperparameters largely around efficiency. We found that the Discrete Fourier Transform is fastest if the window size is close to a multiple of 2. We used a small number of mel spectrogram bins (64) because we found that at that threshold, we could reconstruct the original sequence at an acceptable level using the Griffin-Lim algorithm, [53] which itself might be a *lower bound* on quality as neural methods trained for this purpose have been shown to do better [115].

In our implementation, we compute the spectrogram for an entire video segment (5 seconds) at once; this is of size 64 mel bins by 192 windows. During pretraining, we perform what is effectively a 'random crop' over the spectrogram: we extract three sequential $64x60$ subspectrograms, for each audio subsegment. We constrain them to not overlap, which means that 12 (random) windows are held out.

We note that our Audio Encoder AST is quite different from the one proposed by [47]. Though it operates over spectrograms, we opted for a linear '1-dimensional' layout rather than a two-dimensional (image-like) one. We also did not pretrain our audio encoder on any supervised data (they used ImageNet and found, perhaps surprisingly, that it helped initialize the model). We used a patch size of 64 mel bins by 2 windows; the resulting (1D) sequence is of size 30. After adding a CLS token, the result is a sequence of length **31.**

As we note in the main text, we apply attention pooling afterwards (for all elements except the CLS token), pooling by a factor of five to resize the length-30 sequence to a length of 6 'audio tokens.'

c. Text Span Encoder: We operate on spans that are at most of length 15, with an additional CLS token. Its length is thus **16**.

d. Joint encoder. Let L be the number of text or pooled audio tokens given to the model per segment, on average; we set L=20. Let T be the number of video segments. Then, the joint model's sequence length is $T \times (L + W/32 \times H/32)$. Our total sequence length was thus **640**.

To better adapt our model to downstream tasks – particularly single-image tasks like VCR [126], where past work tends to use a resolution much higher than 192x320, after pretraining, we performed FixRes pretraining (for one epoch on RESERVE-B, and one half epoch on RESERVE-L [108].[10] Here, we trained the model on larger images – simultaneously on 288x512 widescreen images (18 patches by 32 patches), and on 384x384 square images (24 patches on each side). The joint encoder, correspondingly, uses a sequence length of 1312.

During 10 epochs of pretraining, we used a cosine decay of the learning rate down to 0.02 its maximum. During FixRes pretraining afterwards, we warmed up the learning rate to 0.02x its peak, over the first 1/5th of an epoch, and afterwards used a cosine schedule to anneal it towards 0.

Model	GFlops, from Image Encoder	GFlops, from Joint Encoder	GFlops, Total	VCR Q→AR Acc(%)
UNITER-Base[21]	1766	28	1794	58.2
UNITER-Large[21]	1767	99	1867	62.8
MERLOT [128]	236	67	303	65.1
RESERVE-B	99	46	146	62.6
RESERVE-L	176	165	341	**71.5**

Table 7: Efficiency metrics of our model versus others, measured in terms of (giga) floating point operations required to process a single image, question, and answer candidate on VCR. We compare with the overall VCR performance on the combined Q→AR metric. Our RESERVE family of models are significantly more efficient than prior work, with RESERVE-L being roughly on par with MERLOT [128] in terms of FLOPs, yet improving accuracy by over 6%.

B.3. Efficiency metrics of our model

In Table 7, we report efficiency metrics of RESERVE, versus others. We calculate these metrics in the context of scoring a single VCR question and answer candidate. This requires encoding one image, and using 128 tokens for each question and answer combined (for all models). We compare against a UNITER [21], which is a representative VisualBERT style model, along with MERLOT [128]. Our models are far more efficient in terms of FLOPs, with RESERVE-L being roughly on par with MERLOT, yet outperforming it by 6% in terms of VCR accuracy. We discuss key differences below:

a. **UNITER**. We note that UNITER, like other VisualBERT models, uses a supervised object detection backbone [5]. This processes images using a ResNet 101 model [57], at a resolution of 600x800; the final ResNet 'C4' block

[10] We had intended to do a full epoch for RESERVE-L, but our job got preempted, and the loss seemed to have already converged.

is applied densely over the entire image to obtain object-detection potentials everywhere in the image. Both factors greatly increase the FLOPs count.

When computing UNITER's FLOPs count, we exclude operations like non-max suppression, which is an operation that is difficult to implement (and thus whose FLOP count might vary significantly depending on implementation). Our FLOPs count is thus a lower-bound. 36 detection regions are extracted, which is why the 'joint encoder' for UNITER is smaller than the equivalents for MERLOT and RESERVE.

b. MERLOT. This model has two key differences versus our RESERVE. First, it uses a larger image resolution for VCR: 384x704, versus our 288x512. Second, it uses a hybrid ViT-ResNet50 backbone for encoding images. The backbone here is lighter weight than the object detection backbone of UNITER (in particular, the final 'C4' block is removed), and thus, as shown in Table 7, though it uses more FLOPs than does our RESERVE-L, it uses far fewer FLOPs than UNITER.

We choose flops as our primary comparison metric as past work shows that it is one of the key factors in model scaling [66, 34]. Parameters are arguably more fungible. For instance, in text-only representation learning, ALBERT [72] demonstrates that it is possible to *tie parameters* together at all layers of a BERT-like transformer, reducing parameters by an order of magnitude (while not modifying compute), with a minimal performance drop. We did not do this for this work, as we wanted to use a more 'vanilla' Transformer architecture; however, it suggests that representation learning models with hundreds of millions of parameters might be *FLOPs bound* as opposed to *parameter-bound*.

Nonetheless, UNITER-Base has 154 million parameters, though some are frozen (86 million from their Transformer, 23 million from the word embedding layer, and then 44 million from their object detector [5]). UNITER-Large has 378 million parameters (303 from their Transformer, 31 million from word embeddings, and 44 million from the same object detector. Meanwhile, MERLOT has 223M parameters. Versus our RESERVE-B, 14 million extra parameters are due to a larger vocabulary, and 10 million parameters are due to a ResNet50 encoder – but these parameters have a disproportionate impact in FLOPs count.

B.4. Full model hyperparameters

In Table 8, we present full hyperparameters for our model. Among other details, we used AdamW as our optimizer, with $\beta_2 = 0.98$ and $\epsilon = 1e-6$. We increased the learning rate linearly to its peak value (4e-4 for RESERVE-B, 3e-4 for RESERVE-L) over 3750 steps ($\frac{1}{20}$th of an epoch). Our number of warmup steps is lower than many other pretraining work; we note that all of our contrastive objectives involve

		Base	Large
Audio size	Sample rate	22050 Hz	
	FFT hop length	588 samples	
	FFT window size	1536	
	Mel bins	64	
	Subsegment length	60 hops, (≈1.6 sec)	
	Patch size	64 mels × 2hops	
	Pooling ratio	5	
	Final size	6 tokens	
Image	ViT patch size	16	
	Pretraining size	192 × 320	
	Res-adaptation size	288×512 and 384×384	
	Pooling window	2 × 2	
Text	Max. span length	15	
	Mean span length	5.5	
Joint sizes	N video segments	16	
	video segment groups	2 (each with 8 segments)	
	Pretraining seq. length	640 (160 text&pooled audio; 480 pooled vision)	
	Res-adapted seq. length	1312 (160 text&pooled audio; 1152 pooled vision)	
Batch sizes	Videos	1024	
	# Frames (for matching)	16384	
	Masking rate	25% (of subsegments)	
	Text spans	49152	
	Audio spans	49152	
architecture	Hidden size	768	1024
	Num attention heads	12	16
	Size per head	64	
	Rotary size (per head)	32	
	Vision num layers	12	24
	Audio num layers	12	
	Text-span num layers	4	
	Joint num layers	12	24
optimizer	Peak learning rate	4e-4	3e-4
	Weight decay	0.1	
	AdamW β_2	0.98	
	AdamW ϵ	1e-6	
	Warmup steps	3750	
	Training steps	750k (+ 75k for res. adaptation)	
	Training epochs	10 (+ 1 for res. adaptation)	
	σ Maximum scale	100.0	
	Pretraining compute	TPU v3-512 for 16 days	TPU v3-512 for 5 days

Table 8: Architecture details, and pretraining hyperparameters, for both model sizes.

		Base	Large
VCR	Batch Size	32	
	Training Epochs	5	
	Image Size	288×512	
	Learning Rates Tried	1e-5, **2e-5**, 3e-5	**8e-6**, 1e-5, 1.2e-5
	Learning Rate	2e-5	8e-6
TVQA	Batch Size	32	
	Training Epochs	3	
	Image Size	288×512	
	Learning Rates Tried	**5e-6**, 1e-5	**5e-6**, 1e-5
	Learning Rate	5e-6	
Kinetics-600	Batch Size	64	
	Training Epochs	15	
	Image Size	288×512	
	Learning Rate	1e-5	5e-6
	Data Augmentation	From [2]	

Table 9: Hyperparameters for finetuning on downstream tasks. Note that for Kinetics-600, we tried to mimic VATT's setup [2], including adopting their training-epoch regime and their data augmentation strategies. Our data augmentation strategies were much simpler for VCR and TVQA (random cropping, and for VCR sometimes horizontally flipping the image); we suspect that our VCR/TVQA results could be made higher if data augmentation was further explored.

learning a σ parameter, which functions as a secondary 'warmup.'

We did not use gradient clipping. We trained and evaluated in 16-bit bfloat16 precision wherever we could – casting all gradients to that precision as well, and saving the AdamW running mean and variance to be 16-bit as well. A few times during pretraining RESERVE-L, we found that some values in gradients would be NaN. We addressed this by always setting NaN values to be 0. This seemed to address the symptoms of training instability – though sometimes the training loss would spike to roughly around the same loss as random initialization, it always converged back to *slightly better* than it was before the spike. We are not currently sure why this happens.

B.5. Speed improvements during pretraining

We made several high-level algorithmic and engineering implementations to our implementation, which made pretraining run faster, and that we discuss here.

Duplicated video copies. As mentioned in the main text, we create two copies per each video – allowing us to learn separately *how to handle audio as an input* as well as *how to learn from audio*. We chose this in part because copying a video *does not* increase the total compute requried by a factor of two. Instead:

1. We use the image and audio encoders, to encode the underlying video frames and audio clips only once (for the two video copies), and then duplicate the encodings; this is far more efficient than encoding them both separately from scratch.

2. For the two video copies, we sampled two disjoint sets of masks (for which audio and text subsegments are replaced with `MASK`) at a 25% rate. This increases the pool of negative samples for contrastive learning, again increasing training efficiency.

Reducing memory usage. The memory usage of our Transformer implementation scales quadratically with sequence length, which could pose a problem since we operate on sequences of videos. We split the video into two groups of 8 segments, and encode each group separately by the joint encoder.

Vectorization. We vectorize all joint transformer inputs together into a single call. During this vectorization, we also encode the transcript (for the transcript-frame matching objective).

We note that this vectorization is incompatible with the Mask LM variant proposed by MERLOT [128]. In this variant, which the authors called 'attention masking,' two transformer calls must happen *sequentially* – **first**, a language only encoder must encode the inputs and mark down (what is presumably) visually-grounded tokens; **second**, these tokens are masked for the joint encoder. We found that such an objective was unnecessary when pretraining under our contrastive span approach, which in turn enabled more efficient pretraining.

We discuss the exact pretraining data formatting technique that we used in the next section.

C. Pretraining Data Formatting: alignment and masking

In this section, we discuss how we turn a video $\mathcal{V}$ into a (masked) list of segments $\{\boldsymbol{s}_t\}$ for pretraining.

Recall that each segment contains a video frame $\boldsymbol{v}_t$, ASR tokens $\boldsymbol{w}_t$, and audio $\boldsymbol{a}_t$. We generate the list of segments by iterating through the video with a 5-second sliding window.[11]

Audio and text subsegments for masking. We want audio to be used in part as a *target* for contrastive prediction. However, during early exploration we found that 5 seconds of audio could correspond to many BPE tokens; roughly 15 on average. We use past work in language modeling as a guide [64, 93] and wanted an average span length of around 5 tokens. To get this, we split each audio segment

[11] Sometimes there are long 'pauses' in videos where nothing gets said. When this happens – if two segments in a row have fewer than 8 BPE tokens – we merge them 90% of the time, in effect 'fast-forwarding' the audio and still extracting a frame from the middle. We do this at most twice, so the total length is at most 15 seconds here (in effect, a 'playback' rate of 1x, 2x, or 3x). In roughly 90% of cases, the segments are 5 seconds of length.

into three equal *subsegments*, each with a duration of 1.66 seconds. We can then perform masked language modeling at the *aligned subsegment level*, where we mask out the text corresponding to an audio subsegment, and have the model (contrastively) predict the masked-out span of text, as well as the corresponding span of audio. We use a masking rate of 25%, which means that a quarter of the subsegments will be corrupted and replaced by a `MASK` token.

In theory, splitting the videos into (masked) segments ought to be straightforward. However, the key challenge that we ran into is that **the YouTube caption timing information is unreliable**. Problems might arise when we perform pretraining with both audio and text, on misaligned data. Suppose the model is given audio in segment $\boldsymbol{s}_{t-1}$ that ends with somebody saying the word 'pasta.' If the alignment between audio and text is off, the model might be able to cheat the desired task by simply predicting the word 'pasta' for segment $\boldsymbol{s}_t$ – thereby turning the challenging masked-prediction task into an easier speech recognition task; we discuss this in more detail in Appendix C.1.

One way of addressing the timing issue would be to run our own ASR model over all videos, but we chose not to do this due to computational expense. Instead, we adopted two complementary strategies. First, we trained a lighweight regressor to refine the timing information (C.2); second, we mask audio and text conservatively, to minimize alignment errors (C.3). Finally, we discuss how we combine everything efficiently (in a vectorized way) in C.4.

C.1. YouTube Caption Timings

YouTube provides automatically generated captions for accessibility purposes, which include timing information on each word. In the subtitle encoding that we used (`vtt`), each word w contains a single timestamp t which corresponds to when the word should flash on-screen. The timings are *mostly* accurate, but we found two key issues:

a. First, they show up on average roughly 0.1 seconds before each word is spoken, which we suspect might be for usability purposes (perhaps so that while the viewer is reading the caption, they hear the word).

b. Second, with a single timestamp t for each word, it is difficult to infer about pauses. For each word w, we can use its timestamp t, and the timestamps of adjacent words, to loosely infer an interval $[t'_s, t'_e]$ around when the word is said. However, the interval is not tight. We can only infer that the word is being actively spoken for some subinterval $[t_s, t_e]$ such that $t'_s \leq t_s \leq t_e \leq t'_e$.[12] This can lead to high absolute error (in terms of a difference between timesteps), when pauses occur. For example, suppose a speaker says a word, and then pauses. The interval given by the subtitles, $[t'_s, t'_e]$, might be rather large (possibly a few seconds), even though the actual word was spoken for a fraction of that time.

C.2. Refining timing information

We trained a simple multilayer perceptron regressor to correct the timing information of YouTube transcripts. For data, we used 2000 videos with transcripts from YT-Temporal-180M, and also used Google Cloud's (highest quality, paid) ASR service to transcribe them. After aligning the words for these transcripts, this gave us tuples of the YouTube ASR word $\boldsymbol{w}$, its provided interval $[t'_s, t'_e]$, and the 'ground truth' interval $[t_s, t_e]$.[13] Our modeling objective was then to predict the desired offsets with respect to the provided interval: $\delta_s = t_s - t'_s$ and $\delta_e = t_e - t'_e$. We took a feature based approach.

For each input (w, t'_s, t'_e), we used as features:

i. the length of w in characters,
ii. the length of w in BPE tokens,
iii. whether w is uppercase or not,
iv. the number of vowels in w,
v. the number of punctuation characters in w,
vi. the value of $t'_e - t'_s$.

We provided these features as input to the model, as well as the corresponding features for the next word, and the previous word. We z-normalized all features and used a two-layer multilayer perceptron, with a hidden size of 32 and RELU activations. We used a tanh activation at the end to bound the regression. The final predictions for δ_s (analogously for δ_e) were then given by the following equation:

$$\delta_s = c \tanh(\mathbf{w} \cdot \mathbf{h} + b_1) + b_2 \tag{3}$$

where $\mathbf{h}$ is the hidden state, and with learnable parameters c, $\mathbf{w}$, b_1, and b_2. The learned bounds mean that, no matter what the input, the model will never predict an offset of above $c + b_2$ (of which it learned for both parameters $c \approx 0.2$ and $b_2 \approx 0.11$, so the offsets can never be above 0.3 seconds). We trained our lightweight regression model using an L^1 loss, and used it to correct the timing on all of the transcripts.

C.3. Handling worst-case scenarios in masking, when alignment isn't perfect

The regressor that we described reduces the average timing error of a transcript, as a preprocessing step, but it is not perfect. Thankfully, however, we find that most of the remaining alignment errors are *single* words that are slightly misaligned. For instance, for three words w_t, w_{t+1}, w_{t+2},

[12] Note that this is compounded with the first problem, the ground truth interval $[t_s, t_e]$ might not be fully contained in the provided interval $[t'_s, t'_e]$ due to 'captions being shown before audio', the error here is typically small though (0.1 seconds).

[13] When matching YouTube ASR to Google Cloud's ASR, we skipped words without an 'exact-match' alignment, as well as words that were over 0.25 seconds apart (i.e., where either $\delta_s > 0.25$ or $\delta_e > 0.25$

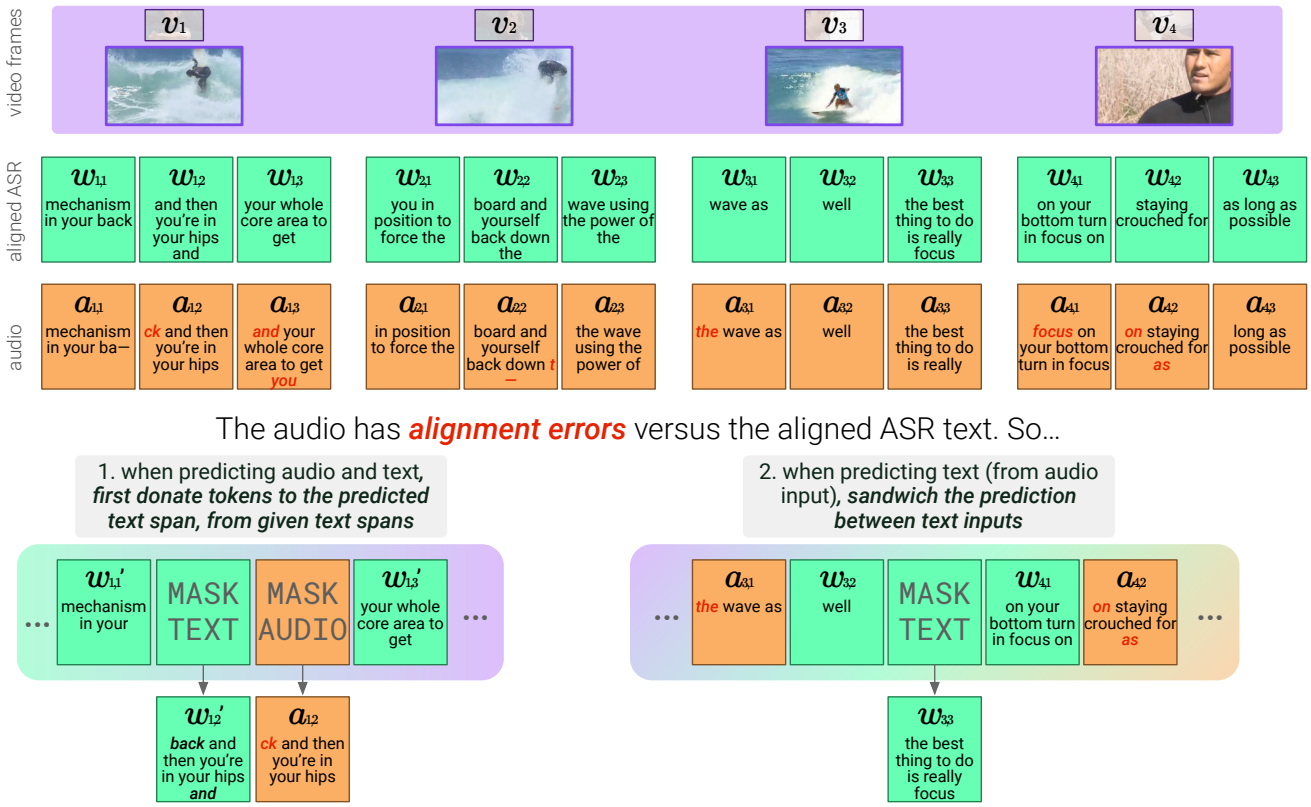

Figure 7: An overview of our masking strategy for dealing with sequences of video frames, ASR, and audio. We have noisy timing information for each word, so we can align the ASR text with audio spans of 1.6 seconds each, using three sub-segments of audio and text for each video frame. However, there exist **alignment errors** between the ASR and audio sub-segments – certain words (and sub-words) have phonemes that are are in the wrong segment (like '***back***' in $\boldsymbol{w}_{1,1}$ is only partially said in the first sub-segment; the 'k' sound is said in the second. When audio is only a target, we address these by 'donating' tokens to predicted spans. When audio is only provided as input, we address this by sandwiching 'mask' tokens between text input (so alignment does not 'bleed' over).

the audio corresponding to the time interval around w_t might contain sound from w_{t+1} being spoken, but rarely w_{t+2}. We suspect this is primarily due to the difficulty inferring pauses: by definition, no other word can be said in a pause, so the errors are local.

We present a high level approach for masking audio and text, that in turn addresses these alignment issues (making it difficult for models to cheat). A diagram is in Figure 7.

Recall that in our framework, we only either go from 'vision and text → text and audio' (VT→TA), or, 'vision, text, and audio → text' (VTA→T). One of the reasons we did this is to avoid allowing a model to cheat by performing speaker identification (or even 'microphone identification'), which might be feasible if audio was given to the joint model as input. We can handle the two cases separately:

a. **Vision and text → text and audio** (VT→TA). Here, the text as input (to the joint encoder) might overlap with the audio we are trying to predict. Our solution here is thus to **donate nearby tokens** from the predicted span, to the input. Let the span that we are trying to predict (and that we will 'mask out') have a start time of t_s and an ending time of t_e. If the final token in the previous text span, if any, has a timestamp of greater than $t_s - 0.125$, we move it to the predicted span; likewise, if the first token in the next text span has a timestamp of less than $t_e + 0.125$, we move it to the predicted span as well.

b. **Vision, text, and audio → text** (VTA→T). In this prediction task, models are given information from all modalities as input, and must predict masked-out text spans. Note that models are only given a single 'speech' modality – either text, or audio – at each timestep. What this means is that we can carefully choose *which input subsegments* to turn into 'audio subsegments,' and which to turn into 'text subsegments.' Our strategy is, given a masked out subsegment, to turn 80% of adjacent subsegments into 'text subsegments.'

We give an illustration of this in Figure 7, part 2. Here the word '***focus***' is part of $\boldsymbol{a_{4,1}}$ but also $\boldsymbol{w_{3,3}}$). This might make $\boldsymbol{w_{3,3}}$) overly easy to predict, if we gave the model $\boldsymbol{a_{4,1}}$ as input. Our solution is thus to give the model text from $\boldsymbol{w_{3,2}}$) and from $\boldsymbol{w_{4,1}}$) as input; we are guaranteed that there is no misalignment overlap here between input and prediction spans. All of the other subsegments (not adjacent to one of the 25% that we mask out) will be provided as audio.

C.4. Putting it all together, along with web text

Finally, we discuss how we combine the various masking approaches into the prediction tasks outlined in the main text.

Each video has $N = 16$ video segments, and three subsegments of audio or text spans per segment. We consider two sub-problems for this video sequence:

i. in VT→TA, vision and text are provided as input, and the model must predict masked-out text and audio. These are done on top of separately-encoded MASK tokens and MASKAUDIO tokens, to enable the model to learn different predictions for each modality over two separate transformer 'columns.'

ii. In VTA→T, vision, text and audio are provided as input, and models must predict masked-out text. Here, we use the term 'predict' as a shorthand for our contrastive objective – in which a model must match a context (a jointly-encoded MASK) to the exact missing span in question, where many negative contexts and spans are provided.

We use a masking rate of 25% for audio and text subsegments, and there are 3 subsegments per segment. This means that a single video instance gives us 48×0.25=12 masked-out spans of text, for each of VT→TA and VTA→T, so 24 in total (as we use disjoint masked-out subsegments). Likewise, it gives us 12 masked-out spans of audio. If we scaled these to the whole batch of 1024 videos, we would have 12k audio span options and 24k text span options. This might suffice, but scaling up the pool of candidates boosts performance in a contrastive setting, as suggested from prior work (e.g. [92]), and as our ablations (Table 1) support as well. Thus, we do the following:

a. Text candidates. We scale up the text candidates by simultaneously training the model on web text, from The Pile [40]. The joint encoder – which can handle pooled video, pooled audio, and BPE-encoded text – is simultaneously given a sequence of web text, for each video that we have. By performing the span-contrastive objective with this piece of web text as well, we can not only teach the model about written (as opposed to spoken) language, but we can scale up the set of text candidates as well.

Let each web-text sequence be of length L. We first divide it into fake regions that 'look like' the text subsegments in length. We do this by calculating the empirical length distribution of the text subsegments, and then using this (categorical) distribution to sample a sequence of sub-segment lengths $\ell_1, \ldots, \ell_K$.[14] We clip the sampled sequence, such that $\sum_i \ell_i = L$.

Next, we mask the fake subsegments. During pretraining, we use text sequences of length $L = 800$, but a model sequence length of only 640. Because we are masking spans and not individual tokens, the text sequences 'shrink' when we mask them. We extract exactly 38 masked-out spans, which corresponds to around 25% of total text.

Finally, we combine the target spans that we took from the webtext sequence, with the target spans from the video. We note that sometimes – especially in a video – text spans might be empty. Not every 1.6 second slice of a video has someone speaking. We thus try to not use these empty spans in our contrastive objective. For each video (which is paired with text for implementation reasons) we select the 'best' 48 text spans out of the (38+24) options – penalizing empty spans, and choosing spans from videos 4x as often.

These 'best 48' text spans, as well as the pooled contexts that they were paired with, will be used in the contrastive objective. Aggregating over the entire batch of 1024 videos (and 1024 web text sequences), this gives us 49152 text spans as candidates, for the all-pairs symmetric softmax between text spans and contexts.

b. Audio candidates. For each video, we note that we have exactly 12 pooled MASKAUDIO tokens, where the model is trying to predict the corresponding audio span. One option would be be to just use those 12 corresponding audio spans as the targets, aggregate these over the batch, and do a symmetric-cross-entropy loss.

However, we can do even better *for free*. Note that for the VTA→T direction, we might have to encode many of the audio spans *anyways*, using the lower level audio encoder (which simultaneously extracts a CLS representation and a sequence-level pooled representation). To simplify implementation, we encode *all 48 audio spans* per video. We can use these audio spans as candidates.

Thus, we do the following when computing the loss over audio prediction. We aggregate all 12288 *contexts* from the MASKAUDIO tokens in the batch, and we aggregate all 49152 candidate audio spans. We perform an all-pairs dot product between these two sets, and use it to compute a symmetric cross-entropy loss over both directions. We did not encounter any trouble using the same temperature for both directions (even though for one direction, there

[14] The empirical distribution for each length, in order from a length of 1 to 15, is [0.03, 0.05, 0.08, 0.11, 0.13, 0.13, 0.12, 0.10, 0.07, 0.05, 0.03, 0.02, 0.01, 0.006, 0.003].

are 12288 options, and for the other, there are 49152).

The combination of these design decisions provide more 'hard negatives' for the model during training. We also found that they worked well to reduce wasted computation on a TPU. For each video, the joint transformer uses one $L = 640$ length sequence for transcript-frame matching, two length-L sequences for the VT→TA direction (as we break it up into two groups of 8 frames each), two length L sequences for the VTA→T direction, and finally one length-L sequence of text. These sequences can all be vectorized together, and the total batch size is $6\times$ the number of videos. This is helpful because using an even-numbered batch size reduces wasted computation on a TPU.

D. Downstream Task Implementation Details

In this section, we present information for how we adapted RESERVE on downstream tasks.

D.1. Setup for finetuned tasks

For adapting RESERVE in a finetuned setting, we take the following approach. We use a linear warmup of the learning rate over the first half of the first epoch, with a linear decay thereafter to 0. To find the learning rate, we did a small grid search generally centered around 1e-5. Our full hyperparameters are shown in Table 8.

When finetuning (and pretraining), we did not use any dropout to make implementation simpler. Instead, as a way to apply regularization, we used the same L_2 penalty as in pretraining (a weight decay of 0.1), but with respect to the *pretrained* weights. This idea was used in [118] among other works, and although it often tends to underperform dropout [73], it is simple to implement.

D.1.1 Visual Commonsense Reasoning

As mentioned in the main text, VCR considers two subtasks: $Q\rightarrow A$, where models are given a question and must choose the right answer given four options; and $QA\rightarrow R$, where models are given a question (and the right answer) and must select the right rationale.

In our setup for this task, we treat it as a four-way classification problem, extracting a single score from each answer or rationale candidate. An example $Q\rightarrow A$ is:

`What is going to happen next? answer: person2 is going to say how cute person4's children are. MASK`

An example $QA\rightarrow R$:

`What is going to happen next? person2 is going to say how cute person4's children are. rationale: It looks like person4 is showing the photo to person2, and person2 will want to be polite. MASK`

We extract representations from the MASK position (which are of dimension d_h), score them with a newly-initialized $d_h \times 1$ weight matrix, and optimize scores with softmax-cross entropy.

Both VCR subtasks use only a single image. We also followed past work in 'drawing on' the provided detection tags to the image [128]. These are unambiguous references to entities that are then referred to in the question, answer, and rationale. For example, text might reference a '`person1`', which corresponds to an image region. When drawing on these detection tags, we do so in a deterministic way – for example, '`person1`' always gets the same box color. We determine the box color by hashing the object's ID (in this case, 'person1') and using that to determine the hue. The model learns the connection between boxes with different hues, and the names, during finetuning.

We randomly flip images left or right, so long as there is no instance of the word 'left' or 'right' in the question, answer, or rationale candidates. We did no other data augmentation (other than randomly resizing images to between 100% to 110% of the network's size).

D.1.2 TVQA

TVQA provides models with a video, a question, and five answer candidates; we represent this as five distinct sequences for the model to score (one per candidate). The version of TVQA that we used also gives models annotations for the *time region* in the video that is being referenced. It is not clear that only using this region would provide enough context to be able to understand what is going on – enough to answer correctly. Thus, for each question, we extract 35 seconds of video around the provided time region. We then provided the model with two numbers corresponding to the time region, relative to the cropped time interval. For example, if the provided timestamp annotation is $[t_0, t_1]$, we use the following region:

$$t_c = \frac{(t_0 + t_1)}{2} \tag{4}$$

$$t_s = t_c - 17.5 \tag{5}$$

$$t_e = t_c + 17.5 \tag{6}$$

The location of $[t_0, t_1]$ in relative coordinates is then:

$$t_0^r = \frac{t_0 - t_s}{t_e - t_s} \tag{7}$$

$$t_1^r = \frac{t_1 - t_s}{t_e - t_s} \tag{8}$$

We provide models with t_0^r and t_1^r, multiplied by 100 and casted to an integer. Thus, an example TVQA instance might look like:

`1 to 28 What is Janice Holding on to after Chandler sends Joey to his room? Chandler's tie. MASK[subtitles or audio]`

This text input corresponds to the first 'segment' of a video; to it we append subtitles (or audio representations) from seven segments from the provided TVQA video (with accompanying frames).

D.1.3 Kinetics-600

We evaluate RESERVE on Activity Recognition over the Kinetics-600 dataset [19]. Here, the model has to classify a short 10-second video clip into a mutually-exclusive set of 600 categories, like 'assembling bicycle' or 'alligator wrestling'. We consider performing this task in a finetuned setting, so as to better compare to prior work. We format each example by extracting 4 video frames from the clip (sampled uniformly), and extracting 6 audio subsegments (totalling 10 seconds of audio). The model processes these inputs along with a `MASK` token, where we extract a vector representation. We initialize the 600-way classification layer with the activations of our Text Span Encoder, over the names of the 600 categories.

We finetune the model jointly over two settings: a setting where audio is provided, and a setting where no audio is provided, to allow us to investigate both settings. We tried to closely follow VATT's finetuning approach [2], including their exact data augmentation settings. We used a batch size of 64 videos (that we process simultaneously 'with audio' and 'without audio'). We used the same image augmentation code as VATT [2], and finetuned for 15 epochs. We used a learning rate of 5e-6 for RESERVE-L and 1e-5 for RESERVE-B.

D.2. Setup and prompting for Zero-shot tasks

Here, we discuss how we set up various tasks for RESERVE in a fully zero-shot setting. In addition to evaluating RESERVE, we also evaluate CLIP [92] in the same zero-shot setting. CLIP is not pretrained on videos, and it cannot jointly encode text. For each task, we construct CLIP's label space by taking our prompt and substituting in each possible answer option. We average together the logits over all frames, and take a softmax, giving us a distribution over the task-specific label space.

D.2.1 Zero-shot Action Anticipation on EPIC-Kitchens

We study the task of action anticipation from the EPIC-Kitchens dataset [26], a large egocentric video dataset with 700 unscripted and untrimmed videos of cooking activities. In action anticipation, a model must predict a *future action* that comes τ_a seconds after a given video clip. The observed segments are of arbitrary length; we follow prior work [26] and set $\tau_a = 1$.

The model tries to choose the correct *noun* and *verb* that happens next, given a list of predefined options for each. We report results on each category using the class-mean top-5 recall.

		Overall			Unseen Kitchen			Tail Classes		
	Model	Verb	Noun	Act	Verb	Noun	Act	Verb	Noun	Act
Validation	RULSTM [38]	27.8	30.8	14.0	**28.8**	**27.2**	**14.2**	19.8	22.0	11.1
	AVT+ (TSN) [46]	25.5	31.8	14.8	25.5	23.6	11.5	18.5	25.8	12.6
	AVT+ [46]	**28.2**	**32.0**	**15.9**	19.5	23.9	11.9	**21.1**	**25.8**	**14.1**
	Chance	6.4	2.0	0.2	14.4	2.9	0.5	1.6	0.2	0.1
	CLIP (VIT-B/16) [92]	13.3	14.5	2.0	12.3	8.4	2.1	14.3	14.3	1.7
	CLIP (RN50x16) [92]	16.5	12.8	2.2	13.4	7.0	1.2	17.1	12.6	2.5
	RESERVE-B	17.9	15.6	2.7	11.0	15.7	4.4	18.0	12.7	2.0
	RESERVE-L	15.6	19.3	4.5	14.1	18.4	3.4	14.7	18.5	4.4
	RESERVE-B (+audio)	20.9	17.5	3.7	15.5	20.1	4.3	20.7	14.5	3.2
	RESERVE-L (+audio)	**23.2**	**23.7**	**4.8**	**20.3**	**21.0**	**5.9**	**22.7**	**21.6**	**4.0**
Test	RULSTM [38]	25.3	26.7	11.2	19.4	26.9	**9.7**	17.6	16.0	7.9
	AVT+ [46]	**25.6**	**28.8**	**12.6**	20.9	22.3	8.8	19.0	22.0	**10.1**
	RESERVE-L (+audio)	24.0	25.5	5.8	**22.7**	**26.4**	7.0	**23.7**	**24.2**	4.7

Table 10: RESERVE gets competitive results on EPIC Kitchen Action Anticipation challenge with zero-shot, over methods from prior work.

Zero-shot inference approach. We directly evaluate the pretrained RESERVE on action anticipation to verify the knowledge learned during pre-training. All prior work reported on the official leaderboard use supervision from the in-domain training set, which we do not use at all [46, 38].

For each action segment, we sample at most $N = 8$ image frames and their associated audio, with fixed time interval $t = 2.0$ preceding it and ending τ_a seconds before the start of the action. We append a `MASK` token as the sole text input (at the last frame, after audio is optionally included).[15] We create short phrases out of all candidate nouns and verbs, and use that as our label space to simultaneously predict them both. We compute the score for each verb and noun independently by averaging their scores, over all labels for which they appear.

Results. We show the full zero-shot action anticipation results in Table 10. We also show our results on the test set here for our best performing model (RESERVE-L, with audio provided). It gets competitive results on verb and noun prediction – with only 1.6% and 3.3% lower compared to the challenge winner method AVT+ [46], which is fully supervised and use additional object-level annotations. On Unseen Kitchen and Tail Classes, our model **outperforms AVT+** on noun and verb. Overall, audio significantly improves the results – RESERVE-L (+audio) outperforms RESERVE-L with an average 3.0%, which suggests that it is useful for this task.

D.2.2 Zero-shot Situated Reasoning

Next, we evaluate on situated reasoning (STAR) [119] which requires the model to capture the knowledge from surrounding situations and perform reasoning accordingly. STAR

[15] We were unable to find a better text based prompt than this, as we found that they often biased the model towards linguistically relevant words; however, we suspect that such a prompt does exist.

dataset includes four types of questions, including interaction, sequence, prediction, and feasibility. A model is given a video clip, a templated question, and 4 answer choices.

Zero-shot inference approach. For each video clip, we sample $N = 8$ image frames uniformly from the video, we also optionally include the video's sound.

To reduce domain shift between YouTube data – where people don't typically ask visual questions, and where ASR typically does not insert question marks – we convert the question-answer pair into a statement. We did so using the question-answer templates provided by the author, with the answer replaced by a `MASK`. For example, *"Q: What did the person do with the bottle? – A: Put down."* will be converted to *"The person* `MASK` *the bottle."*.

We put the converted statement into the first frame and use the four candidate answers as a unique label space (that differs from example to example). Like with EPIC-Kitchens, we also evaluate how much audio can help by masking the audio inputs.

Results. We show our zero-shot STAR results in Table 5 in the main text. Our base model outperforms all supervised prior work by 3.7%. The model with audio performs better, with average 1.1% improvement. Interestingly, RESERVE-L is worse than RESERVE-B, we suspect the reason is RESERVE-L is sensitive to grammar details. Given the previous example, we note that while 'Put down' is a valid answer that might make sense both semantically and syntactically, a different answer 'pick up' might be flagged by some English speakers as being ungrammatical: the instantiated template would then be 'the person pick up the bottle.' We noticed instances of the larger model paying greater attention to these syntax-level details, even though they were not the focus of the task. It does suggest, however, that additional prompting (or label space augmentation) could resolve these issues and increase performance even further.

D.2.3 Zero-shot LSMDC

We evaluate our model on Movie Fill-in-the-Blank [96, 82] task, which based on descriptive audio description for the visually impaired. Given a movie clip and an aligned description with a blank in it, the task is to fill in the blank with the correct word. Following [82], we report prediction accuracy in test set of 30,354 examples from 10K movie clips.

Zero-shot Inference approach. We sample $N = 8$ video segments uniformly over the movie clip, and extract the audio and middle frame of each segment. We replace the 'blank' token in each description with a `MASK` token, and provide it (as text-based input) to the model at its final segment. For the other segments, we optionally provide the model with audio; for all segments, we provide the associated image frame. We use the vocabulary set in the LSMDC dataset as our label space (for what the 'missing word' might be).

Results. Our results are shown in Table 5 in the main text. Our model obtains 31% when audio is included, which outperforms human text-only performance (30.2 %) [82], predicted by human annotators. A supervised LSTM obtains 34.4% in this text-only setting [82] which suggests that there is a certain textual bias in this task, which our model cannot learn (as it is zero-shot). This also suggests that state-of-the-art supervised models exploit patterns in this vocabulary distribution.

Without such an advantage, our model performs well, outperforming CLIP (2%) by a large margin. This suggests that jointly reasoning over both the visual situation, and the linguistic context of the provided sentence, is helpful for zero-shot performance on LSMDC fill-in-the-blank.

D.2.4 Zero-shot MSRVTTQA

Finally, we evaluate our model on MSR VTT-QA, a question-answering task over videos [120]. We provide a model with $N = 8$ video segments sampled uniformly from the video clip, and extract an image from each one. For the first seven segments, we optionally include audio extracted from that point; at the last segment, we insert a converted version of the question, along with a `MASK`. We compare the similarity of that hidden state to the top 2000 most common answers, similar to past work [128].

Similar to STAR, we convert the questions into statements to minimize drift away from the pretraining distribution. We use GPT3 prompted with several examples for this. Our exact prompt is the following:

```
    Input:  what is a car being driven through?
Output:  a car is being driven through _.
Input:  who are running across screen?
Output:  _ are running across screen.
Input:  when is a girl performing?
Output:  a girl is performing at _.
Input:  what is a cartoon doing?
Output:  a cartoon is _.
Input:  how many women talk in a bedroom?
Output:  _ women talk in a bedroom.
Input:  what a man playing while dancing with others?
Output:  a man is playing _ while dancing with others.
Input:  where is a flag hoisted?
Output:  a flag is hoisted in _.
Input:  who talks to another man on the couch?
Output:  _ talks to another man on the couch.
Input:  what does a teenage girl try to get at a public restroom?
Output:  a teenage girl tries to get _ at a public restroom.
Input:  when do the models walk as the audience watches?
Output:  the models walk as the audience watches at _.
```

```
Input:  what shows a person killing animals in a green forest?
Output:  _ shows a person killing animals in a green forest.
Input:  who does a man ask to go on a date?
Output:  a man asks _ to go on a date.
Input:  what are three people sitting on?
Output:  three people are sitting on _.
Input:  ${question}
Output:
```

Then, given a new question `${question}`, GPT3 generates a converted output, wherein we can replace it's underscore with a `MASK`. GPT3 works well at this conversion, though sometimes it generates a sentence where inserting the 'correct answer' feels gramatically strange. For example, the question 'how many women talk in a bedroom?' suggests any integer might be a reasonable answer. On the other hand, '_ women talk in a bedroom' implies that 'one' is not a valid answer (since 'women' is plural). We note that the errors caused by this conversion technique are specific to English grammar, and so if such a question-conversion approach was done in other languages, there could be more (or less) errors that directly result.

Our results are shown in Table 5. Of note, our model through automatic question-conversion outperforms Just Ask [123], which performs an analogous (supervised-guided) question conversion on all its YouTube transcripts, before pretraining. Our model also outperforms CLIP, which cannot naturally handle dynamic situations.

E. Dataset Collection

In this section, we discussed how we curated data for YT-Temporal-1B. We had several goals in mind. We wanted to use only public-facing data, which motivated our choice of YouTube as it is a public platform *that users understand is public* [65]. We wanted to use this platform to examine to what extent we can learn multimodal neural script knowledge from web data alone.

Our data collection strategy in this work was informed by past work, notably MERLOT [128]. That paper found that increasing the diversity and scale of a video corpus both allowed for better learned representations. At the same time, the data collected by MERLOT (YT-Temporal-180M) has issues. Of note, the authors' scraping strategies – to prioritize *monetized content* – also led to a lot of U.S. local news being in that corpus (roughly 30% of all data). Local news might be problematic to learn from, particularly in that quantity, due to its numerous biases (e.g. racist coverage on 'crime' [45, 31, 30, 58]). Our goal was to expand the dataset in both diversity and size to 20 million videos, while having *less local news* and without scraping private content.

High level approach. We adopt a similar dataset collection strategy as in MERLOT [128]. In the first phase, we identify a candidate set of videos ID to download. In the second phase, we open each video ID in YouTube and apply several filtering steps that go from inexpensive to expensive. The filtering steps allow us to exit early and possibly avoid downloading the video if the video seems unsuitable for our purpose from the title, description, and captions alone.

For a Datasheet [41], please see the MERLOT paper [128].

E.1. Candidate video IDs

For MERLOT's YT-Temporal-180M, the bulk of the video IDs were identified by applying breadth-first-search on YouTube channels from HowTo100M [85] and VLOG [36]. Each channel often links to other channels, and given a channel it is inexpensive to obtain a list of all its videos using the youtube-dl Python package.

In this paper, we considered numerous approaches to search for diverse, visually grounded videos. We ended up using an approach where we used YouTube's recommended videos algorithm to suggest similar videos to YT-Temporal-180M. We went through all non-news and non-sports videos YT-Temporal-180M, and opened each video up in YouTube. For each other video that YouTube recommended, we retrieved its channel ID – giving us access to not just that video, but all other videos. This approach yielded 2 million channels, with 200 million videos among them.

E.2. Filtering video IDs by channel

Given this (large) list of channels, each with many videos, we took steps to filter it further. We used the python `cld3` library to remove channels whose titles might not be in English. We then finetuned, and used, a language model to identify channels likely to have visually grounded videos, which we describe next.

In more detail, we selected 2000 videos, and asked workers on Mechanical Turk to rate their level of groundedness, their genre, and whether they had explicit content or not. The questions we asked are shown in Figure 8. We annotated 2k videos under this schema, and trained a model to predict the annotations given video metadata.

For model training, we used a slightly different setting to what we gave the crowdworkers. We trained a model to predict the labels, given a formatted list of 5 video titles from the same channel. During training, we made the weak-supervision assumption that all videos from a channel have exactly the same rating (as the video we annotated). This enabled us to collect 84k examples from our 2k annotations. The model we chose was T5-base model [93], which generates the labels left-to-right in text form (and which we converted automatically to a structured representation).

We then used this model to identify channels that seem especially promising. For each channel with at least 5 videos, we randomly sampled 8 sets of length-5 videos, and used the finetuned T5 model to classify them. We filtered out any channel that had at least 25% of likely non-English or

${VIDEO}

Q1. How would you describe the role of English speech in the video?

a. This video doesn't have spoken English, or if it does, it's irrelevant to what's going on in the video.
b. This video has English speech that describes, or adds onto, the visual content.

Q2. Select at least one genres of the video:

a. Gaming
b. News
c. How-to
d. Chatting
e. Sports
f. Music
g. Movies / Drama
h. Documentary
i. Miscellaneous

Q3. Select if any of the following are true:

a. A variety of objects are interacted with.
b. A variety of actions are performed.
c. A variety of scenes are performed.
d. This video is a slideshow.
e. This video contains racist or sexist content..

Figure 8: Video annotation. We had workers on Mechanical Turk annotate 2000 videos in our dataset with this questionnaire, allowing us to then train a model to identify suitable channels for our purpose.

irrelevant-English videos, any channel that had at least 25% of slideshows, and any channel that likely had racist or sexist content.

One side benefit of this model is that it allowed us to estimate our videos' genre breakdown before downloading them. We found 1% Gaming videos, 11% News videos, 20% How-To videos, 20% 'chatting' videos, 5% sports videos, 5% Music videos, 3% Movies/Drama videos, 4% Documentary videos, and 31% Miscellaneous. The Gaming videos were then filtered out.

We used the classification model to create a budget for how many videos to download from each channel; with the aim to download more videos from likely more-grounded channels. Using the answers to Q3 (from Figure 8), we gave each channel 1 point for likely having 'a variety of objects', 2 points for 'a variety of actions', and 0.5 points for 'a variety of scenes.' We subtracted 3 points if it was likely to be a slideshow. (Likely-racist or sexist channels were already filtered out.) We then z-normalized and softmaxed the channel scores, and used the result as the channel-level budgets. Any channel with an aggregate 'interestingness' score of 1 standard deviation above the mean would then have a budget of 8x larger than the mean. We clipped the channel-level budgets to include at most 500 videos per channel.

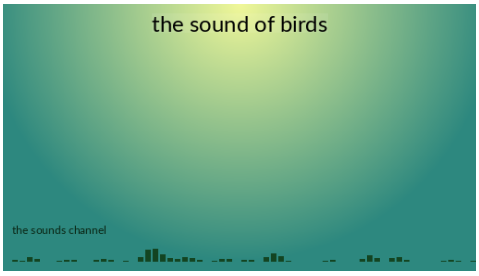

Figure 9: An image prompt used in zero-shot audio classification. Here, "the sound of" is always inserted, and the word "birds" is one of the labels in ESC50 [90]. We consider one image prompt for each label in ESC50 (or whichever dataset we are using).

This process (finally!) gave us 30 million YouTube video IDs that were likely to be high-quality.

E.3. Filtering videos from their metadata

Last, we filtered and downloaded these videos using a filtering approach similar to [128]. We first retrieved the video metadata and used it to filter out 'gaming' videos. We then retrieved the video's transcript, and filtered out any video without a 'dense' span of spoken words – defined as an interval of 30 seconds where at least 50 words are spoken. Additionally, we used the Python package `cld3` to filter out any transcript with a probability of less than 80% of being English. Last, we used a hidden feature in the YouTube API to download four thumbnails of the video. Using the image classification model from [128], we filtered out videos whose four thumbnails had an average cosine similarity of above 85%, or that contained fewer than 1 object from COCO.

Unlike [128], we did not use a sequence-to-sequence model to 'translate' spoken text to text that appears more stylistically like written English (i.e., by adding capitalization and punctuation, and removing filler words).

F. Additional Experiments and Exploration

In this section, we briefly include additional experiments, showcasing our model's performance on specific tasks that do not necessarily require multimodal script knowledge.

F.1. Zero-shot Audio classification

We evaluate RESERVE on the task of zero-shot audio classification, to study to what extent its learned audio representations can directly predict text-based labels. We conduct this evaluation on environmental sounds from ESC50 [90], urban sounds from US8K [98], and (as part of the privacy-minded exploration in Appendix A) celebrity voices from VoxCeleb2 [87].

Model	Prompting	Accuracy (%) ESC50	US8K	VoxCeleb2
AudioClip		68.6	68.8	
RESERVE-L	Text-only.	41.6	60.2	10.8
	Image-only.	42.8	54.3	**13.3**
	Image and text.	**52.2**	**62.3**	9.6

Table 11: Zero-shot audio classification accuracies (%) on ESC50 [90], US8K [98], and VoxCeleb2 [87]. We compare our model with AudioClip [54], which was pretrained on supervised data from AudioSet [43]. Our RESERVE performs well across the board, especially when given *both the image and the text* as a prompt – demonstrating its OCR capability.

We consider the format where we encode an audio input into a `CLS` level representation, and retrieve the most-similar label given a set of encoded options. We encode the audio input with our encoder, which takes in as input audio clips of length at most 1.6 seconds. For shorter audio clips (like many sounds in ESC50), we repeat them in time until their length is at least 1.6 seconds. For longer audio clips, we encode multiple `CLS` representations and then average the resulting vectors.

We consider the following ways to encode the labels:

a. **Text-only**. Inspired by the prompt 'a photo of', which is used in CLIP's zero-shot image classification task [92], we give RESERVE's joint encoder a blank image, with associated tokens `the sound of ${label}`. We do this once for each label, giving us a single 'target' vector for each possible label in the dataset.

b. **Image-only**. Inspired by YouTube videos of sound effects[16], we created image-only prompts that suggest a sound (of the target class) is playing in the background. An example is shown in Figure 9. We encode each image with our joint encoder, and do this once for each label.

We note that for VoxCeleb2, we use face images of celebrities rather than this image-based prompt, due to our interest in exploring whether models can perform person-level recognition due to the privacy issue (Appendix A.1.1).

c. **Image and text.** Here, we combine both of the above options: encoding one input for each label, using both the image and text prompt.

For each prompt, we append the token '`MASKAUDIO`' and extract the hidden state from there, as our final representation for that label.

We present our results in Table 11. The results show, possibly surprisingly, that RESERVE can perform optical character recognition over image prompts like Figure 9 – given just the image, its accuracy on ESC50 is higher than given just text. Its accuracy on ESC50 and US8K improves further when given both an image and text.

These results are slightly different for VoxCeleb2, which emphasizes long-tail recognition of people – something that might be more encyclopedic than semantic, and that we did not wish to optimize in this work. There, when given an image of a celebrity's face, it demonstrates some capacity at linking it with one of their audio clips – a capacity that *decreases* if prompted with additional text. We suspect that this is due to interpreting the given text as spoken, for example, *Justin Bieber himself saying* 'the sound of Justin Bieber.' On all celebrities, RESERVE struggles versus recognition-focused models like CLIP [92] (Appendix A.1.1).

Overall, our model displays strong audio understanding ability. In comparison, AudioCLIP [54] (which is supervised on human-annotated labels from AudioSet [43]), performs 16% higher on ESC50, and 6.4% higher on US8K.

F.2. Additional Qualitative Analysis

In Figure 10, we include an additional figure of examples, of the same format as Figure 5. The examples are chosen randomly – not by how much RESERVE improved at retrieving their audio or text spans over the course of training.

[16]For instance, youtu.be/VmgKryu4__k.

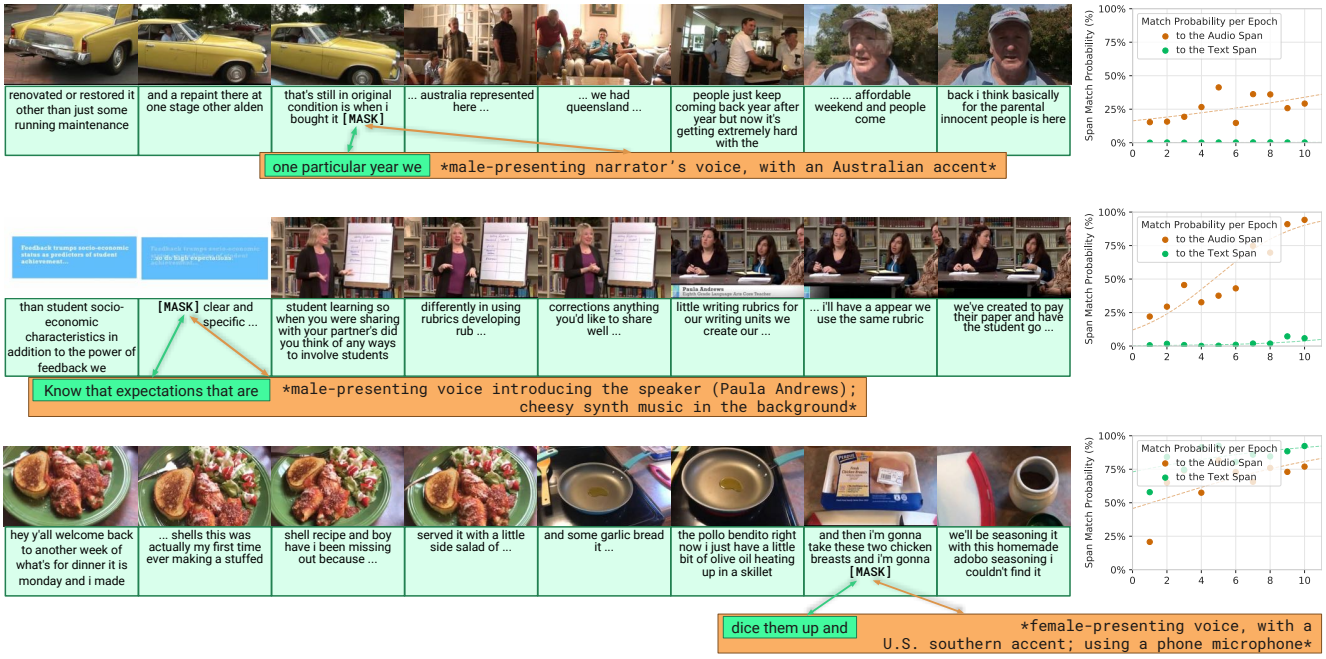

Figure 10: **MASKed audio self-supervision on different examples**. Similar to Figure 5, we show predictions from RESERVE-B over the course of pretraining. Match performance increases over time. The audio prediction in the first row is perhaps made easier by the speaker's australian accent. The audio prediction in the second row is perhaps easier due to the lecture-video setting. In the third row, both audio and text span prediction improves, with text being slightly favored in the end. This might be in part because of the truncation we do on audio (Section C.3) – the audio span is shorter than the text span of 'dice them up and' so as to not leak information, making prediction more challenging.