

# Deepfake Detection Using Spatiotemporal Methods and Vision-Language Models

Timothy Tong  
ttong@scu.edu  
Santa Clara University  
Santa Clara, California, USA

David Anastasiu  
danastasiu@scu.edu  
Santa Clara University  
Santa Clara, California, USA

Yuhong Liu  
yhliu@scu.edu  
Santa Clara University  
Santa Clara, California, USA

## Abstract

From elections, to wars, to people’s personal lives, deepfakes have become ever more present, blurring the distinction between reality and fiction. Many deep learning-based deepfake detection methods lack generalizability. They often overfit to deepfake generation techniques in their training distribution, and fail to detect deepfake generation techniques outside of their training distribution. One approach to solve the generalization problem has been to use vision-language models, such as the pretrained CLIP model, to extract features that generalize across different deepfake generation techniques, including diffusion and GAN images. However, most CLIP approaches only consider image-level features for detection, such as artifacts from blending or generative models, rather than video-level features such as flicking or temporal inconsistencies. The importance of capturing both spatial and temporal information is demonstrated by two state-of-the-art detection methods: AltFreezing, which uses a 3D CNN and an alternating weight freezing strategy to train both spatial and temporal weights; and TALL, which uses a Swin-Transformer and a 2D thumbnail layout to capture spatial information within frames and temporal information across consecutive frames. To compete with the state-of-the-art in video deepfake detection, we propose three deepfake detection methods that integrate temporal information with the spatial information that CLIP is already able to capture. The first combines CLIP with a transformer trained from scratch. The second combines CLIP with TimeSformer, a transformer-based architecture built for video understanding. The third uses X-CLIP, a variant of CLIP for video understanding, with its multiframe integration transformer.

## CCS Concepts

• **Computing methodologies** → **Computer vision; Machine learning**; • **Security and privacy**;

## Keywords

Deepfake Detection, Spatiotemporal Modeling, Vision–Language Models, CLIP, TimeSformer, X-CLIP, Transformer Networks, Video Forensics, Temporal Inconsistencies

## ACM Reference Format:

Timothy Tong, David Anastasiu, and Yuhong Liu. 2025. Deepfake Detection Using Spatiotemporal Methods and Vision-Language Models. In *Proceedings of KDD Undergraduate and Master’s Consortium (KDD-UMC ’25)*. ACM, New York, NY, USA, 8 pages. <https://doi.org/XXXXXXXX.XXXXXXX>

## 1 Introduction

Deepfakes have a pervasive impact on all aspects of our lives. The impact of deepfakes can range from the global to personal scale. In times of war, bad actors can use deepfakes to manipulate public perceptions. A 2022 NPR report [1] discusses a deepfake video in which Zelensky orders his troops to surrender during the currently ongoing conflict between Ukraine and Russia. Bad actors can similarly use deepfakes for political disinformation. In 2024, Reuters [23] reported robocalls going around in New Hampshire, where deepfaked audio of Biden urged people not to vote in the Democratic primary. Deepfakes can even damage personal lives. ABC News [20] reported in January 2025 that a high school senior in Sydney created fake social media profiles of his classmates, and then posted deepfake porn of said classmates. Meanwhile, deepfake technology is constantly improving. In December 2024, OpenAI [17] released their text-to-video platform, Sora, allowing users to create realistic synthetic videos, and remix, blend, or extend different video clips. In May 2025, Google DeepMind [9] released their own text-to-video platform, Veo 3, allowing users to generate videos with audio. With the rise of such generation tools available for anyone to use, companies have also delved into the realm of detection to deter bad actors. In 2022, VentureBeat [8] discussed Intel’s Fake-Catcher, which achieves 96% accuracy by focusing on the blood flow of deepfake faces, but is not accessible to the public. Most recently, in conjunction with the release of Veo 3, Google DeepMind [10] announced SynthID, which embeds detectable watermarks into AI generated content, from image, audio, text, to even video, in order to prevent misinformation. However, until the mass adoption of watermarking in all generation tools, it seems that the more useful approach is to train deep learning models to do the detection. Due to these challenges with company-proposed solutions, open-source deepfake detection models appear to be a better alternative for public use. However, they vary in quality and face a common issue of poor generalizability. They will overfit to deepfake generation techniques in their training distribution, and fail to detect deepfake generation techniques outside of their training distribution.

One approach to solve the generalization problem has been to use vision-language models, such as the pretrained CLIP model, to extract features that generalize across different deepfake generation techniques. Most CLIP approaches only consider image-level features for detection, such as artifacts from blending or generative models, rather than video-level features such as flicking or

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](mailto:permissions@acm.org).  
*KDD-UMC ’25, Toronto, ON, Canada*

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
ACM ISBN 978-1-4503-XXXX-X/2025/08  
<https://doi.org/XXXXXXXX.XXXXXXX>

temporal inconsistencies. The importance of capturing both spatial and temporal information is demonstrated by two state-of-the-art detection methods: AltFreezing [24], which uses a 3D CNN and an alternating weight freezing strategy to train both spatial and temporal weights; and TALL [25], which uses a Swin-Transformer and a 2D thumbnail layout to capture spatial information within frames and temporal information across consecutive frames.

To address the poor generalizability of deepfake detection methods, and allow the public to better defend against misinformation at the geopolitical, national, and personal level, it is imperative that we investigate methods that combine the advantages of foundation vision-language models and spatiotemporal methods. We propose three methods that integrate temporal information with the spatial information that CLIP is already able to capture: (1) CLIP with a transformer trained from scratch, (2) CLIP with TimeSformer [2], and (3) X-CLIP [15], a variant of CLIP with video understanding capabilities, with its multiframe integration transformer.

## 2 Related Works

### 2.1 Spatiotemporal Methods

Wang et al. [24] identify a trend in the literature that most deepfake detection methods work only on the image-level. In other words, they focus on spatial artifacts, such as those related to blending or generative models. These methods are unable to detect deepfakes at the video-level, that is, they are unable to detect temporal artifacts, such as incoherence or flicking. While temporal methods can capture deepfakes that have been generated frame by frame, they fail to capture deepfakes that have been generated coherently at the video-level. These coherent deepfakes, however, have spatial artifact weaknesses.

In order to develop a more generalizable deepfake detection method that captures both types of artifacts, a common approach is to use a spatiotemporal model, such as a 3D CNN. However, Wang et al. argue that such a model tends to overly rely on spatial artifacts because they are easier to learn, and so it fails to learn temporal artifacts. Hence, the authors introduce their AltFreezing training strategy, which divides the weights in a spatiotemporal model such as a 3D CNN into spatial and temporal weights. During training, the spatial weights are frozen for a certain number of iterations, in order to let the model focus on learning temporal features. Then, the temporal weights are frozen for a certain number of iterations, in order to let the model learn spatial features. The authors tune the ratio of freezing iterations between spatial and temporal weights, finding the most effective ratio to be 20:1, where spatial weights are frozen significantly more, to prevent their overreliance. Inspired by their findings, we choose to pursue separate components for spatial and temporal weights in our own methods, to ensure that our methods do not overly rely on spatial features. To prevent overreliance on spatial features, we place the temporal component after the spatial component, theorizing that later layers have a stronger gradient signal due to a shorter backpropagation path.

Given the better capabilities of vision transformers, Xu et al. [25] opt to use a vision transformer instead of a 3D CNN for spatiotemporal modeling. They introduce a thumbnail layout (TALL), dividing each video into eight segments, sampling four consecutive frames at random from each segment, then resizing and rearranging these

four frames as subimages in a  $2 \times 2$  thumbnail with the same size as the original frames. Despite the loss of spatial information, the authors argue that this arrangement helps make their training more efficient in terms of computation and memory. They combine this strategy with an existing Swin-Transformer-based deepfake detection method to create TALL-Swin, which uses a shifted window that computes self-attention within subimages in the thumbnail, as well as across multiple subimages in the thumbnail. Hence, their method is able to capture local, intraframe dependencies, as well as global, interframe dependencies. They find that for deepfake detection, video-level methods that capture temporal information perform better than methods that focus on only image-level information. Given the effectiveness of TALL and video-level detection, we decide it would be worthwhile to look into CLIP as an alternative vision transformer, and to explore different strategies for a video-level detection method using CLIP.

### 2.2 CLIP-Based Approaches

Liu et al. [14] call attention to the recent surge of detection approaches that apply vision-language models as feature extractors, namely, CLIP, a pretrained model that consists of two encoders to enable multimodal capabilities: a vision transformer which generates image embeddings from the input, and a separate transformer which generates textual embeddings from the input.

Ojha et al. [16] point out a key weakness of the existing fake image detection paradigm of training a deep learning neural network for binary classification between real and fake images. They explain that existing networks learn a decision boundary based on the fingerprints of the specific generative model seen during training, as it is easier to learn compared to the features that make up a real image. Therefore, the detector will classify fake images from another generative family as real, since they will not contain similar fingerprints previously seen during training. The authors propose a solution of using a feature space not learned for the purpose of deepfake detection. Specifically, they propose using the CLIP ViT-L/14 feature space, keeping the CLIP model as a frozen backbone, and using nearest neighbor and linear probing methods. For the nearest neighbor method, the authors create a feature space from fake and real images, then classify test images based on their cosine distance to fake or real images. For the linear probing method, the authors train a single linear layer appended to the CLIP backbone for binary classification on labeled fake and real images. The authors find that both approaches achieve better generalization compared to existing deep learning methods, such that when only exposed to real and GAN images beforehand, their methods are able to better detect diffusion images at test time. They argue the essential component of the CLIP image encoder that makes their method successful is that rather than trained on a small dataset, the encoder is trained at the Internet scale. From their results, they conclude that their linear probing strategy achieves similar performance to their nearest neighbor approach, while remaining more computationally and memory efficient. It is due to the advantages of their method that we append a linear layer stack for binary classification in each of our proposed approaches. They also report that training CLIP for binary classification leads to the same poor generalization as in existing methods, however, when experimenting with our own

methods, we find that fine-tuning some of CLIP’s layers boosts performance.

According to Cozzolino et al. [5], traditional CLIP-based deepfake detection methods assume that a large training dataset in the same image domain as the deepfakes is required for good results. The authors present a technique where CLIP only needs a few images in order to achieve generalizable and robust deepfake detection. Using CLIP to extract feature embeddings from real and fake images, and training an SVM classifier to differentiate between them, the authors demonstrate that with only ten fake and ten real images in the training set, their classifier achieves high performance on GAN and diffusion images. They furthermore show high generalizability across data domains: the CLIP-based classifier achieves stable and high AUC when trained on data generated using latent diffusion, and evaluated on various GAN, diffusion, and commercial image generators. Even when the images in the training set are noisy, the CLIP-based method maintains high AUC. This connects to another reason we choose to pursue CLIP-based approaches, which is that we are unable to reproduce several results reported in the literature for our baselines, despite trying datasets preprocessed by other authors, as well as by ourselves. However, we are able to achieve our best baseline results using CLIP, possibly due to its high adaptability to different data. The CLIP-based method by Cozzolino et al. achieves the highest average AUCs of 88.5% with only 20,000 images and 89.4% when also training on preprocessed images. The authors discover that it is more difficult for deepfake attackers to circumvent the CLIP-based method, because CLIP seems to be extracting higher level features that are independent from the forgery traces that many baseline detection models rely on. Thus, another reason we use CLIP is because of its practical benefit in the real world to be able to withstand attacks that hide or tamper forgery traces.

The previous studies use CLIP to detect fake images, but not fake videos, and specifically, deepfake videos that impersonate people’s faces. A recent study in March 2025 from Han et al. [11] explores this use case. They find that when using a completely fine-tuned CLIP model for deepfake detection, the model tends to focus on general facial regions that do not provide useful information for detection, limiting the method’s generalizability. In their work, they freeze the CLIP ViT-L/14 image encoder. When passing in the frames of a deepfake video, each layer of CLIP extracts attention attributes for key, query, and value, as well as patch embeddings, and sends them to a decoder the authors propose, consisting of a spatial module and a temporal module. The spatial module computes cross attention, using its weights as queries, the attention attributes as keys, and the patch embeddings as values, producing a layer-wise spatial embedding. A facial component guidance loss helps align the spatial weights to attention attributes for the nose, eyes, skin, and lips, which are key regions for deepfake detection, rather than generic regions. Separately, the temporal module computes self-attention across time for the attention attributes of the same patch location in different frames, then applies a series of convolution and reshaping operations to combine temporal information across patches and representations and integrate spatial context, to produce a layerwise temporal embedding. Finally, the spatial and temporal embeddings are aggregated across all layers, and sent to three linear heads: a temporal head uses the temporal embeddings,

a spatial head uses the spatial embeddings, and the spatial-temporal head uses the concatenated temporal and spatial embeddings. The average prediction across the heads is the final prediction. While the authors freeze CLIP, training their decoder modules and linear heads, we investigate CLIP as the primary learner for the spatial features of deepfakes, directly fine-tuning CLIP’s encoder layers. Rather than have layerwise modules, we place our entire temporal component after the CLIP encoder model, so that the temporal component learns separately from the spatial dimension. This is followed by linear layers that predict only on spatiotemporal features. In an ablation study, the authors find that their spatial module provides the greatest boost in video AUC of 5.3%, followed by their temporal module of 3.3%. While they find their largest improvement from spatial learning, we choose to focus on investigating temporal innovations that boost detection performance, and let CLIP handle the spatial dimension.

### 3 Methods

We propose three spatiotemporal methods using vision-language models: CLIP with transformer, CLIP with TimeSformer, and X-CLIP with multiframe integration transformer. Each of these methods uses the power of vision-language models to extract highly generalizable image features for deepfake detection, and also leverages a temporal component to capture deepfake artifacts that appear across multiple frames. Our CLIP with transformer method consists of a CLIP vision model that captures the spatial features of each video frame, and a stack of transformer encoders to temporally pool the spatial information. Our CLIP with TimeSformer method consists of a CLIP vision model that captures the spatial features of each video frame, and a TimeSformer encoder that uses divided space-time attention to pool CLIP’s extracted features. Our X-CLIP with multiframe integration transformer method consists of an X-CLIP vision model that extracts features from each video frame and computes self-attention across the different frames, and a multiframe integration transformer that averages information across the frames. We now cover each individual method in more detail.

#### 3.1 CLIP with Transformer

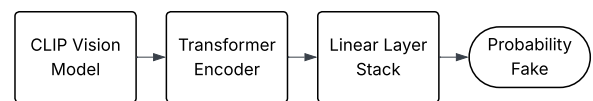


Figure 1: CLIP with transformer architecture.

The first method we propose is a combination of CLIP with a transformer encoder, seen in Figure 1. Our input features are of shape (batch size, clip size, channels, height, width). Clip size is also known as the number of frames in each video clip. We iterate through each of the frames, sending it to our backbone, which is the vision model in CLIP ViT-L/14. When instantiating our backbone, we freeze all parameters, and then unfreeze only the last 8 encoder layers in the vision model, hence, for this method we only fine-tune these last 8 layers. So, we apply our vision backbone to each frame,

just taking the pooler output, which is the CLS token that summarizes the spatial information extracted from each frame [19]. We stack these results together getting a shape (batch size, clip size, embedding dimensions). In this case, the embedding dimensions is 1024. We then apply a linear projection layer, to bring the embedding size down from 1024 to 256, which helps extract higher-level features and reduce redundancy. Then, in front of the stack of features extracted from each frame, we prepend a learnable CLS token. This token learns to summarize the information among the rest of the frames during self-attention. We then add positional encoding to keep track of the order of the frames, since self-attention by itself loses sequence information. Finally, we pass this whole result through 2 transformer-encoder layers. Each encoder layer has a hidden size of 256, 8 heads, 2048 feed-forward dimension, 0.525 dropout, ReLU activation,  $1e-5$  LayerNorm eps, and apply LayerNorm before self-attention and feed-forward stages. From the final result of these encoder layers, we take just the CLS token and feed it to a linear layer with input size 256 and output size 128. We apply batch normalization, ELU activation, and 0.2 dropout. We send the result to another linear layer of input size 128 and output size 64. We then apply batch norm and ELU activation, and 0.05 dropout. We finally feed the result to a final linear layer for classification with input size 64 and output size 2, applying softmax and taking the probability that the video clip is a deepfake as our prediction.

### 3.2 CLIP with TimeSformer

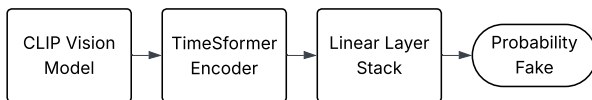


Figure 2: CLIP with TimeSformer architecture.

Next, we investigate combining CLIP with TimeSformer, seen in Figure 2. This method is inspired by Coccomini et al. [4], where they replaced the patch embeddings of TimeSformer with a CNN backbone, allowing them to connect the two models together for deepfake detection. TimeSformer uses divided space-time attention [2]. Each TimeSformer encoder layer contains a phase where it computes self-attention across the temporal dimension, followed by a phase where it computes self-attention across the spatial dimension. Once again, our input feature shape is (batch size, clip size, channels, height, width). We use the vision model of CLIP ViT-L/14, unfreezing all parameters, and use TimeSformer pretrained on Kinetics-600 [3], which is a YouTube video dataset consisting of 600 action classes. We use gradient checkpointing to resolve out-of-GPU-memory issues. For TimeSformer, we freeze all model parameters, then unfreeze just the encoder layers, as well as the final LayerNorm. We also save the CLS token, position embeddings, time embeddings, position dropout, and time dropout from the patch embedding layers. At the start of the forward pass, we reshape our features to have shape (batch size x clip size, channels, height, width). Then we feed these features to our CLIP backbone to extract spatial features from each frame. For this method, we take the last hidden state as our output, which has shape (batch size x clip size, 1 + number of patch tokens, embedding dimension). The embedding

dimension as before is 1024. We remove the CLS token at the front of the patch tokens. The issue now is that we have  $16 \times 16 = 256$  patch tokens from our CLIP model (since our input images are  $224 \times 224$ , and a patch side length is 14, hence we have  $224/14=16$ ). However, TimeSformer’s encoder layers are expecting  $14 \times 14 = 196$  patch tokens (where each patch side length is 16, hence we have  $224/16=14$ ). Therefore, we reshape the output to be (batch size x clip size, 16, 16, embedding dimension). Then, we permute the shape so that it becomes (batch size x clip size, embedding dimension, 16, 16). We apply 2d adaptive average pooling so that the output has new shape (batch size x clip size, embedding dimensions, 14, 14), and we flatten across the last two dimensions and permute so that we get a final shape of (batch size x number of patch tokens x embedding dimension), where the number of patch tokens is now 196. Now, we use a linear layer to project the embedding dimension from 1024 to 768, to get an input shape that TimeSformer is expecting. This process allows us to skip over the patch embedding step of TimeSformer. Then, we repeat the steps as in the original TimeSformer implementation. We first prepend the saved CLS to the front of the patch tokens. Then, we add the saved positional encoding and apply the saved positional dropout. We extract CLS out of the data temporarily, resulting in a shape (batch size x clip size, number of patch tokens, embedding dimension), which we permute and reshape to become (batch size x number of patch tokens, clip size, embedding dimension). We now add the saved time embeddings, apply the saved time dropout, and reshape the output to be (batch size, number of patch tokens x clip size, embedding dimension). Then we prepend the CLS tokens to get a final shape of (batch size, 1 + number of patch tokens x clip size, embedding dimension). We pass this through TimeSformer’s encoder layers, and take the last hidden state, which we apply TimeSformer’s LayerNorm. Then we extract the CLS token at the front of the middle dimension, and pass it to a linear classification layer of input size 768 and output size 2. As before, we apply softmax and take the probability that the video clip is a deepfake as our prediction.

### 3.3 X-CLIP with Multiframe Integration Transformer

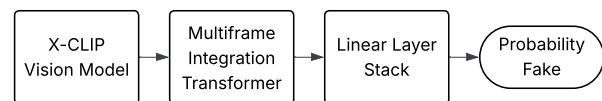


Figure 3: X-CLIP with multiframe integration transformer architecture.

Finally, we investigate using X-CLIP along with its multiframe integration transformer for deepfake detection, seen in Figure 3. X-CLIP [15] is an extension of the CLIP model that enables it to extract features from videos in addition to images and text. The video component of X-CLIP consists of patch embeddings to create patches from each video frame, a cross-frame communication transformer to enable frame tokens to communicate with each other across time, and a multiframe integration transformer to average the information across the video frames. There is very limited research

applying X-CLIP for deepfake detection, and prior to implementing our proposed method, we did not find any deepfake detection research that applies the multiframe integration transformer, as most research only uses the vision model component consisting of the patch embeddings, the vision encoder components from the original CLIP model, and the cross-frame communication transformer. However, afterwards, we found work by Pang et al. [18] where they use X-CLIP’s full model, including the multiframe integration transformer. Our contribution is that we evaluate on the standard benchmarks and metrics in the deepfake detection literature, specifically training and evaluating on a variety of GAN, autoencoder, and graphics-based deepfake videos, using video-level AUC as our performance metric. On the other hand, their paper takes a less common approach of evaluating on only diffusion-generated images, and furthermore uses accuracy as their performance metric. Finally, we essentially perform an ablation study in our work, as one of our baselines uses X-CLIP without its multiframe integration transformer, so we are able to provide more insight regarding its impact on deepfake detection. As before, our input features are (batch size, clip size, channels, height, width). We use the full model of X-CLIP ViT-L/14 as our backbone, applying gradient checkpointing. Similar to TimeSformer, X-CLIP is also pretrained on a YouTube video dataset, this time on Kinetics-400 [12], which has 400 action categories. We apply X-CLIP’s tokenizer with max length padding, truncation, and tokenizer max length on an empty string, and from the resulting tensor extract input ids and an attention mask. Then, we pass our entire input features, along with the input ids and attention mask, to the fully X-CLIP backbone. This enables us to use the full X-CLIP backbone with the multiframe integration transformer, which is separate from the vision model component. We then grab just the pooled video embeddings, which have shape (batch size, embedding dimension), and apply LayerNorm. Then, we feed the result to a linear classification layer with input size 768 and output size 2. Again, we apply softmax and take as our prediction the probability that the video clip is a deepfake.

## 4 Experimental Design

### 4.1 Datasets

In accordance with the baseline papers, for our training dataset, we exclusively use the c23 (high quality, lightly compressed) images from **FaceForensics++ (FF++)** [21], which consists of 1.8 million deepfake images generated from 1000 original YouTube videos using two learning-based methods (DeepFakes and NeuralTextures) and two graphics-based methods (FaceSwap and Face2Face). According to the settings on DeepfakeBench [26], we validate on **Celeb-DF-v2 (CDF-v2)** [13], which consists of 5639 deepfake videos, generated from 590 original YouTube videos of celebrities using a refined version of the DeepFakes method with less color mismatches and temporal flickering, and increased image resolution and mask accuracy. For our testing datasets, besides FF++ and CDF-v2, we evaluate our baselines and proposed solution on five other datasets supported on DeepfakeBench. **Celeb-DF-v1 (CDF-v1)** [13] is the first version of the Celeb-DF dataset that consists of 795 deepfake videos generated from 408 original YouTube videos of celebrities. **DeepfakeDetection (DFD)** [21] consists of 3068 deepfake videos using different deepfake methods found online,

generated from 363 original videos shot in controlled-settings with paid actors. **FaceShifter** [21] applies a two-stage deepfake generation technique on the same 1000 original YouTube videos in FF++, where the first stage uses a GAN-based method to preserve the lighting and resolution of the target image in the final, synthesized image, and the second stage identifies facial occlusions such as shadows, reflections, or glasses covering the face, to refine the image. **Deepfake Detection Challenge Preview (DFDCP)** [7] uses two unknown face swapping methods, method A and method B, to generate 3865 and 144 videos, respectively, from 1052 original videos shot in controlled settings. **Deepfake Detection Challenge (DFDC)** [6], specifically, its public test set of 4704 videos, consists of deepfakes generated from staged videos with paid actors, applying five different face swapping methods that vary in quality. Most of its deepfakes are generated using Deepfake Autoencoder (DFAE), also known as DeepFakes in other datasets; other methods include MM/NN face swap, Neural Talking Heads (NTH), FSGAN, and StyleGAN. Our setup enables us to measure both intra-dataset and cross-dataset performance of the models.

### 4.2 Baseline Methods and Metrics

We investigate four baselines: AltFreezing, TALL, X-CLIP, and CLIP. The former two are discussed in Section 2.1. The latter two use only the model backbone (CLIP or X-CLIP) followed by a linear classifier. See Appendix A.2 for more details.

Following the literature, we report video-level AUC. After the model processes every frame in a video, giving a prediction for each frame, we average the results to get an overall video-level prediction. We repeat over the entire dataset split to get a video-level AUC.

## 5 Experimental Results and Discussion

Our results are shown in Table 1. ViT-B/16 refers to the base model that uses a vision transformer with patch size 16 x 16 and embedding size 768, while ViT-L/14 denotes the larger model that uses a vision transformer with patch size 14 x 14 and embedding size 1024. Furthermore, CLIP ViT-L/14 + T denotes our proposed method that combines CLIP with a transformer, CLIP ViT-L/14 + TF denotes our method that combines CLIP with TimeSformer, and X-CLIP ViT-L/14 + MIT denotes our method that combines X-CLIP with its multiframe integration transformer. We have bolded our proposed methods, and drawn a line of separation in the table where the top half is our baselines, and the bottom half is our proposed methods. We have also bolded the best results from evaluation on each of our datasets. We have additionally provided the last two columns to demonstrate the average performance across all the datasets (including FF++), and the generalization average performance (without FF++).

To address the performance differences across datasets, we refer back to our discussion in Section 4.1 for analysis, but further study is needed. For FF++, the best performing method is X-CLIP ViT-L/14 + MIT, at 99.54% AUC, while the second best performing method is X-CLIP ViT-L/14 alone, at 99.50% AUC. We previously note that X-CLIP is pretrained on YouTube videos, and the FF++ dataset consists of original YouTube videos as well as deepfakes generated from those original videos. It is possible that pretraining on similar or possibly even overlapping data gives the X-CLIP models an

**Table 1: Video-Level AUC (%) from Cross-Dataset Evaluation**

	FF++	CDF-v1	CDF-v2	DFD	FaceShifter	DFDCP	DFDC	Avg	Gen Avg
AltFreezing	96.69	89.21	80.57	70.71	81.45	63.46	64.50	78.08	74.98
TALL	97.62	80.97	86.37	87.19	66.64	78.11	74.44	81.62	78.95
CLIP ViT-B/16	99.30	75.98	83.71	86.50	63.07	84.27	79.68	81.79	78.87
CLIP ViT-L/14	99.39	89.43	<b>90.89</b>	92.33	80.94	88.44	81.85	89.04	87.32
X-CLIP ViT-L/14	99.50	83.73	87.43	96.19	89.69	89.63	82.96	89.88	88.27
<b>CLIP ViT-L/14 + T</b>	96.07	<b>89.58</b>	89.18	90.43	81.62	88.47	81.33	88.10	86.77
<b>CLIP ViT-L/14 + TF</b>	99.36	87.04	88.82	<b>97.08</b>	83.74	<b>92.15</b>	82.49	90.10	88.55
<b>X-CLIP ViT-L/14 + MIT</b>	<b>99.54</b>	88.25	87.90	95.88	<b>90.35</b>	87.84	<b>84.50</b>	<b>90.61</b>	<b>89.12</b>

advantage. However, TimeFormer is also pretrained on YouTube videos, but our CLIP ViT-L/14 + TF method performs slightly worse than CLIP ViT-L/14 alone. Therefore, it is also possible that it is one of the model architecture components of X-CLIP itself that enable it to perform well on this dataset, such as the patch embeddings or the cross-frame communication transformer.

For CDF-v1, the best performing method is CLIP ViT-L/14 + T, at 89.58% AUC, followed by CLIP ViT-L/14, at 89.43% AUC. It is interesting that the generally higher performing methods with TimeFormer and with X-CLIP perform worse here. A similar trend occurs with CDF-v2, except the best performing method is CLIP ViT-L/14, at 90.89% AUC, while the second best performing method is now CLIP ViT-L/14 + T, at 89.18% AUC. Once again, the X-CLIP and TimeFormer methods underperform. This can relate to how both TimeFormer and X-CLIP are pretrained on YouTube videos, while both CDF-v1 and CDF-v2 consist of original YouTube videos of celebrities, as well as deepfakes generated from those videos using a similar version of the DeepFakes generation method. TimeFormer and X-CLIP could be ignoring certain features from these videos that would hint at deepfake manipulation due to their pretrained knowledge. On the other hand, it might be easier for the CLIP ViT-L/14 models to pick up on their deepfake features, having only been trained on image and text.

For DFD, the best performing method is CLIP ViT-L/14 + TF, at 97.08% AUC, followed by X-CLIP ViT-L/14, at 96.19%, and then X-CLIP ViT-L/14 + MIT at 95.88%. We previously note that DFD differs from the previous datasets, as it consists of staged videos using paid actors, rather than original YouTube videos from the wild. The deepfake generation methods are also not publicly stated. It can be argued here that the models' pretrained knowledge of video is generalizable to the staged videos in DFD, allowing for a significant performance boost over the fourth best performing method, CLIP ViT-L/14 alone, at 92.33%.

For FaceShifter, the best performing method is X-CLIP ViT-L/14 + MIT, at 90.35% AUC, followed by X-CLIP ViT-L/14, at 89.69%. As discussed before, FaceShifter aims to create higher quality deepfakes using a two stage synthesis process that preserves lighting and resolution of the target image, and identifies facial occlusions for refinement. Given the much lower performance of the third best performing method, CLIP ViT-L/14 + TF, at 83.74%, we suspect that the model architecture components of X-CLIP, such as the patch embeddings or the cross-frame communication transformer, somehow offer an advantage in detecting these deepfakes.

For DFDCP, the best performing method is CLIP ViT-L/14 + TF, at 92.15% AUC, followed by X-CLIP ViT-L/14, at 89.63%. We refer back to our discussion on DFDCP, specifically that it consists of staged videos, and deepfakes generated from those videos using two face-swapping methods, method A and method B, whose real identities are not publicly known. It can be argued here that the divided space-time attention particular to TimeFormer, rather than just temporal pooling, is offering a significant advantage in terms of detecting the deepfakes by the two unknown methods.

For DFDC, on the other hand, the best performing method is X-CLIP ViT-L/14 + MIT, at 84.50% AUC, followed by X-CLIP ViT-L/14, at 82.96%, and then CLIP ViT-L/14 + TF, at 82.49%. As discussed previously, the majority of DFDC consists of deepfakes generated using the DeepFakes generation method, also called Deepfake Autoencoder (DFAE). Similar versions of the same method are also applied in CDF-v1 and CDF-v2. Indeed, we see that in all three of these datasets, X-CLIP ViT-L/14 + MIT outperforms X-CLIP ViT-L/14 alone, from anywhere between 0.47% to 4.52%, which suggests that the multiframe integration transformer, by averaging information across each video, gives an advantage in detecting autoencoder generated deepfakes. We also see a slight advantage from pretrained knowledge of video, in the TimeFormer and X-CLIP methods, for detecting deepfakes in this dataset.

Overall, our work provides key insights into the benefits of using pretrained models with built-in temporal knowledge of videos, namely, TimeFormer and X-CLIP, to augment the spatial capabilities of CLIP models for deepfake detection. Specifically, our CLIP ViT-L/14 + TF and X-CLIP ViT-L/14 + MIT methods beat our best baseline of X-CLIP ViT-L/14 in overall AUC performance. This also demonstrates the benefit of the multiframe integration transformer in allowing for more stable predictions. When testing on sample videos, all three methods output fake probabilities above 90% for deepfake videos, although our best baseline outputs slightly higher confidence scores. This, however, may indicate our best baseline's greater tendency to predict fake, as we also observe that our best two performing methods are less prone to false positives. For a sample real video, our best performing methods give it a 0.84% to 9.3% chance of being fake, while our best performing baseline gives it a 51.83% chance of being fake. These observations align with our results in Table 1, as our own methods have higher video-level AUC, which means that they are better able to separate between the fake and real video classes, although further study is needed for confirmation.

## References

- [1] Bobby Allyn. 2022. Deepfake video of Zelenskyy could be ‘tip of the iceberg’ in info war, experts warn. *NPR* (March 2022). <https://www.npr.org/2022/03/16/1087062648/deepfake-video-zelenskyy-experts-war-manipulation-ukraine-russia>
- [2] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. 2021. Is Space-Time Attention All You Need for Video Understanding? arXiv:2102.05095 [cs.CV] <https://arxiv.org/abs/2102.05095>
- [3] Joao Carreira, Eric Noland, Andras Banki-Horvath, Chloe Hillier, and Andrew Zisserman. 2018. A Short Note about Kinetics-600. arXiv:1808.01340 [cs.CV] <https://arxiv.org/abs/1808.01340>
- [4] Davide Alessandro Cocomini, Giorgos Kordopatis Zilos, Giuseppe Amato, Roberto Caldelli, Fabrizio Falchi, Symeon Papadopoulos, and Claudio Genaro. 2022. MINTIME: Multi-Identity Size-Invariant Video Deepfake Detection. doi:10.48550/ARXIV.2211.10996
- [5] Davide Cozzolino, Giovanni Poggi, Riccardo Corvi, Matthias Nießner, and Luisa Verdoliva. 2024. Raising the Bar of AI-generated Image Detection with CLIP. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.
- [6] Brian Dolhansky, Joanna Bitton, Ben Pfau, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. 2020. The DeepFake Detection Challenge Dataset. arXiv:2006.07397 [cs.CV]
- [7] Brian Dolhansky, Russ Howes, Ben Pfau, Nicole Baram, and Cristian Canton Ferrer. 2019. The Deepfake Detection Challenge (DFDC) Preview Dataset. arXiv:1910.08854 [cs.CV]
- [8] Sharon Goldman. 2024. Intel unveils real-time deepfake detector, claims 96% accuracy rate. *VentureBeat* (Nov. 2024). <https://venturebeat.com/ai/intel-unveils-real-time-deepfake-detector-claims-96-accuracy-rate/>
- [9] Google DeepMind. 2024. Veo 3: Video, meet audio. Our latest video generation model. <https://deepmind.google/models/veo/>. Accessed 27 May 2025.
- [10] Google DeepMind. 2024. Watermarking AI-generated text and video with SynthID. *Google DeepMind Blog* (May 2024). <https://deepmind.google/discover/blog/watermarking-ai-generated-text-and-video-with-synthid/>
- [11] Yue-Hua Han, Tai-Ming Huang, Kai-Lung Hua, and Jun-Cheng Chen. 2025. Towards More General Video-based Deepfake Detection through Facial Component Guided Adaptation for Foundation Model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [12] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. 2017. The Kinetics Human Action Video Dataset. arXiv:1705.06950 [cs.CV] <https://arxiv.org/abs/1705.06950>
- [13] Yuezun Li, Honggang Qi Xin Yang, Pu Sun, and Siwei Lyu. 2020. Celeb-DF: A Large-scale Challenging Dataset for DeepFake Forensics. In *IEEE Conference on Computer Vision and Patten Recognition (CVPR)*.
- [14] Huan Liu, Zichang Tan, Chuangchuang Tan, Yunchao Wei, Jingdong Wang, and Yao Zhao. 2024. Forgery-aware Adaptive Transformer for Generalizable Synthetic Image Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [15] Yiwei Ma, Guohai Xu, Xiaoshuai Sun, Ming Yan, Ji Zhang, and Rongrong Ji. 2022. X-CLIP: End-to-End Multi-grained Contrastive Learning for Video-Text Retrieval. arXiv:2207.07285 [cs.CV] <https://arxiv.org/abs/2207.07285>
- [16] Utkarsh Ojha, Yuheng Li, and Yong Jae Lee. 2023. Towards Universal Fake Image Detectors that Generalize Across Generative Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [17] OpenAI. 2024. Sora is here. <https://openai.com/index/sora-is-here/>. Accessed 27 May 2025.
- [18] Yan Pang, Yang Zhang, and Tianhao Wang. 2024. VGMSHield: Mitigating Misuse of Video Generative Models. arXiv:2402.13126 [cs.CR] <https://arxiv.org/abs/2402.13126>
- [19] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. arXiv:2103.00020 [cs.CV] <https://arxiv.org/abs/2103.00020>
- [20] Ethan Rix. 2025. Sydney high school senior investigated by police over deepfake pornographic images of female students. *ABC News* (Jan. 2025). <https://www.abc.net.au/news/2025-01-09/sydney-high-school-deepfake-images-other-students/104799338>
- [21] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. 2019. FaceForensics++: Learning to Detect Manipulated Facial Images. In *International Conference on Computer Vision (ICCV)*.
- [22] Santa Clara University WAVE HPC Wiki. n.d. WAVE HPC Technical Specifications. [https://wiki.wave.scu.edu/doku.php?id=wave\\_hpc#wave\\_hpc\\_technical\\_specifications](https://wiki.wave.scu.edu/doku.php?id=wave_hpc#wave_hpc_technical_specifications). Accessed: 26 May 2025.
- [23] David Shepardson. 2024. Consultant fined \$6 million for using AI to fake Biden’s voice in robocalls. *Reuters* (Sept. 2024). <https://www.reuters.com/world/us/fcc-finalizes-6-million-fine-over-ai-generated-biden-robocalls-2024-09-26/>
- [24] Zhendong Wang, Jianmin Bao, Wengang Zhou, Weilun Wang, and Houqiang Li. 2023. AltFreezing for More General Video Face Forgery Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4129–4138.
- [25] Yuting Xu, Jian Liang, Gengyun Jia, Ziming Yang, Yanhao Zhang, and Ran He. 2024. TALL: Thumbnail Layout for Deepfake Video Detection. arXiv:2307.07494 [cs.CV] <https://arxiv.org/abs/2307.07494>
- [26] Zhiyuan Yan, Yong Zhang, Xinhang Yuan, Siwei Lyu, and Baoyuan Wu. 2023. DeepfakeBench: A Comprehensive Benchmark of Deepfake Detection. In *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 4534–4565. [https://proceedings.neurips.cc/paper\\_files/paper/2023/file/0e735e4b4f07de483cbe250130992726-Paper-Datasets\\_and\\_Benchmarks.pdf](https://proceedings.neurips.cc/paper_files/paper/2023/file/0e735e4b4f07de483cbe250130992726-Paper-Datasets_and_Benchmarks.pdf)

## A Supplementary Details

### A.1 Technologies and Environment Setup

We integrate our three deepfake detection methods into the DeepfakeBench repository [26]. This is a benchmark that provides standardized code for training and evaluating deepfake detection models, such as helper functions, loss functions, training and testing code. It also comes with many baseline and state-of-the-art deepfake detection methods already implemented, and preprocessed deepfake video datasets, allowing for fair comparison of methods.

To set up the environment, we clone the DeepfakeBench repository and follow its README file. This includes creating an environment for Python 3.7.2, and running the provided install script, which among other packages installs the CUDA 11.3 build of Torch 1.12.0 and associated torchvision and torchaudio. We also download the preprocessed datasets provided by the benchmark, obtaining 32 extracted faces, masks, and landmarks for each video. The preprocessed data consists of 256 x 256 images of faces obtained from sampling 32 frames per video, detecting the face in each video frame, aligning each face, and then cropping the margins around each face. The last step is necessary so that the model can focus on relevant facial features rather than overfit to the background details. The benchmark provides both RGB and LMDB data, but due to geographical restrictions we are only able to obtain the RGB data, along with their JSON configuration files. We arrange the datasets and JSON files according to the benchmark’s instructions, running the rearrangement script to synchronize data loading across datasets. Finally, to train and evaluate each model, we followed the benchmark’s steps to run their provided scripts, specifying the YAML configuration, datasets, and model weights, if any.

### A.2 Baseline Implementation

DeepfakeBench provides the code for our baselines. The original CLIP baseline uses ViT-B/16. We find that the larger ViT-L/14 variant performs significantly better. Each image is passed through CLIP, and the features are then passed to a linear classifier with 1024 input dimensions, and 2 output dimensions. Softmax generates probabilities for the final classification outcome. The CLIP baseline is trained without using video-level data. Each data sample is a single frame from a video, and each training batch shuffles individual frames from different videos together. The AltFreezing and X-CLIP baselines are trained using video clips of eight frames each, while the TALL baseline is trained using video clips of four frames each. Each sample is a clip of consecutive frames from the same video, so each training batch shuffles different clips from different videos together.

### A.3 Hyperparameters

Baseline hyperparameters are as follows. For AltFreezing baseline, we use train and test batch sizes of 32, sampling 32 frames per video, and making clips of size 8 frames. We use the default data augmentation and mean and std values from DeepfakeBench. We use SGD optimizer with learning rate 0.2, momentum 0.9, and weight decay 5e-4. To align with the original paper, we use a cosine learning rate scheduler, with the minimum learning rate of 0 being

reached after 1000 epochs. We hence train for 1000 epochs, using cross-entropy loss.

For TALL, we apply default DeepfakeBench settings for mask grid size, embedding dimensions, MLP ratio, patch size, window size, depths, number of heads, absolute position embedding, thumbnail rows, drop rate, and drop path rate. We use train and test batch sizes of 64, sampling 32 frames per video, creating clips of 4 frames. We turn off data augmentation, and apply the mean values of 0.485, 0.456, 0.406, and the std values of 0.229, 0.224, 0.225 provided by DeepfakeBench for normalization. We use Adam optimizer with learning rate 2e-5, beta1 0.9, beta2 0.999, eps 1e-8, weight decay 5e-4, and train for 100 epochs using cross-entropy loss. We save the best weights using video-level AUC instead of AUC.

For CLIP ViT-B/16 only, we use a train and test batch size of 32, sampling 32 frames from each video, and creating clips of 8 frames. We apply the default data augmentation from DeepfakeBench. We use Adam optimizer, applying learning rate 2e-5, beta1 0.9, beta2 0.999, eps 1e-8, and weight decay of 1e-3, training for 10 epochs with cross-entropy loss. For CLIP ViT-L/14 only, we use a training batch size and test batch size of 16, using 32 frames from each video. We create clip sizes of 8 from each video. We apply the HuggingFace preprocessor for CLIP ViT-L/14 on each of the frames. We keep the default data augmentation in DeepfakeBench for flip, rotation, blur, brightness, and quality lower and upper bounds. We increase data augmentation for the following: rotate limit to between -15 and 15, blur limit to between 5 and 9, brightness limit to between -0.15 and 0.15, contrast limit to between -0.15 and 0.15. We use the same optimizer settings as for our CLIP ViT-B/16 only baseline, training for 10 epochs, and using cross-entropy for the loss.

For X-CLIP only, we use a train and test batch size of 32, and as before sample 32 frames per video, creating clip sizes of 8 frames. Instead of using a HuggingFace preprocessor, we use the mean and std values that DeepfakeBench provides for X-CLIP: for mean, the values are 0.485, 0.456, 0.406, and for std, the values are 0.229, 0.224, 0.225. We turn off data augmentation, and apply Adam optimizer with learning rate 2e-6, beta1 0.9, beta2 0.999, eps 1e-8, and weight decay 5e-4. We train for 100 epochs, using cross-entropy loss.

Hyperparameters for our proposed methods are as follows. For CLIP with transformer, we use the same settings as our CLIP ViT-L/14 only baseline. For CLIP with TimeSformer, we use the same batch sizes, frame number, clip sizes, and preprocessor as for CLIP with transformer. We turn off data augmentation, and again apply Adam optimizer, with same beta1, beta2, and eps as in CLIP with transformer. However, we find that our best results are when we fine-tune TimeSformer at a higher learning rate of 2e-5, and CLIP at a lower learning rate of 2e-6. We apply a weight decay of 5e-4, and train for 10 epochs, again using cross-entropy loss. For X-CLIP with multiframe integration transformer, we use the same settings as for our X-CLIP only baseline.

### A.4 Hardware

We use Santa Clara University’s High-Performance Computing Center [22] to run single-GPU training and evaluation with NVIDIA Tesla V100, AMD Instinct MI100, and NVIDIA L40S GPUs.