FakeSV: Multimodal Misinformation Detection in COVID19 Short Videos

Team DeepTok

- Zhiwei He(zhiwei_he@brown.edu)
- Jinjia Zhang(jinjia_zhang@brown.edu)
- Xulong Xiao(xulong_xiao@brown.edu)
- Zhangchi Fan(zhangchi_fan@brown.edu)

Introduction

Short video platforms like Douyin and Kuaishou have become major sources of news sharing — but also major breeding grounds for fake news. Existing fake news detection methods suffer from small datasets and insufficient use of multimodal and social context information.
To address this, we reimplemented FakeSV, the largest Chinese fake news short video dataset, including video content, user comments, and publisher profiles.
SVFEND is a multimodal detection model, which is the core component of FakeSV and is capable of:

- Identify important features across different types of media like text, audio, and images.
- Combine content and social information (comments) to strengthen its detection.
- Accurately catches various kinds of fake news videos.

Dataset

Original Dataset (FakeSV):

- Source: Crawled from Douyin (TikTok) and Kuaishou, combined with fact-checking articles.
- Size: 5,538 videos (1,827 fake, 1,827 real, 1,884 debunked) across 738 events.

Features:

- News Content: Video frames, audio, titles, transcripts.
- Social Context: User comments, publisher profiles (verified status, activity metrics).

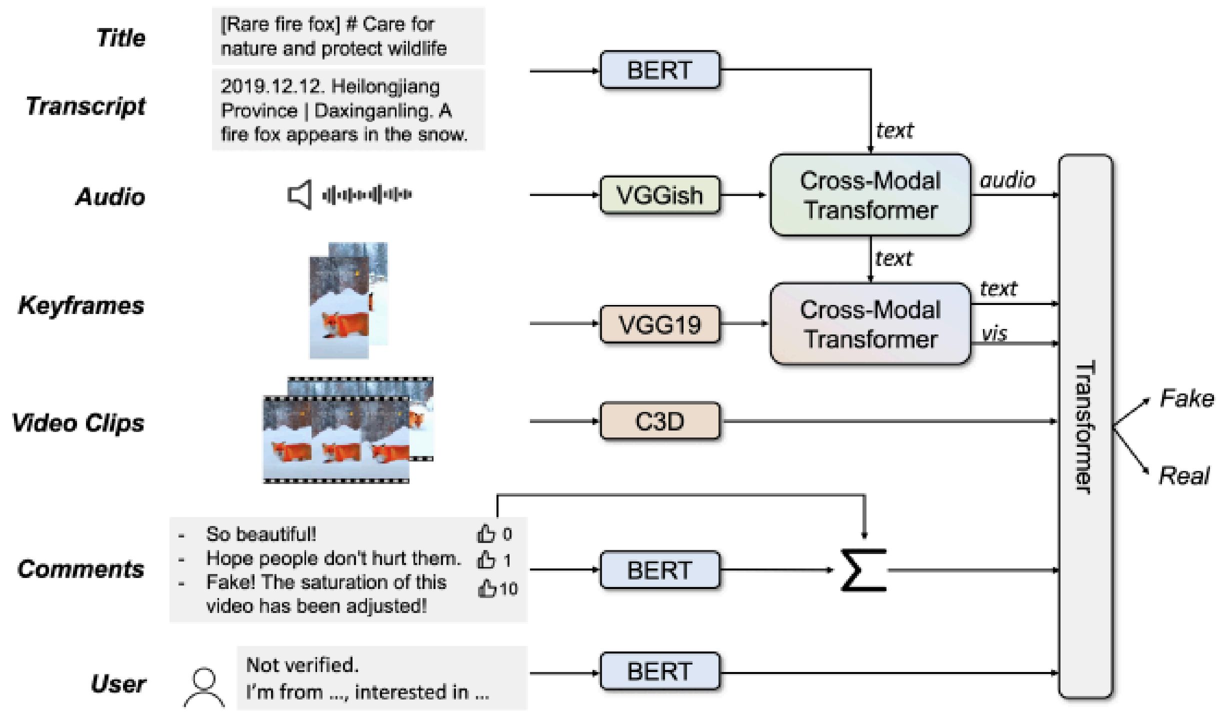
Methodology

Model Architecture (SVFEND):

1. Multimodal Feature Extraction: Extract features from video title + transcript (BERT), audio (VGGish), visual frames (VGG19), video clips (C3D), comments (BERT with like-weighted pooling).
    a. Text: BERT for titles/transcripts.
    b. Audio: VGGish for acoustic features.
    c. Video: VGG19 (frames) and C3D (motion).
    d. Social Context: BERT for user profiles, weighted comment features.
2. CrossModal Fusion: Two crossmodal transformers align and enhance textual, audio, and visual features. Helps spot key information across modalities (e.g., matching speech tone with video frames).
    a. Co-attention transformers to align text, audio, and visual features.
    b. Self-attention to integrate social context.
3. Social Context Fusion: A selfattention transformer fuses multimodal content features with user comments and publisher information for final prediction.

4. Multimodal Feature Fusion and Representation: Aggregate all enhanced features (text, audio, visual, comments, user profile) into a unified representation using a standard self-attention layer.

5. Classification: Fully connected layer with softmax. Pass the final fused feature through a fully connected layer + Softmax to predict whether a video is real or fake, trained using binary cross-entropy loss.


Title
[Rare fire fox] # Care for nature and protect wildlife
Transcript
2019.12.12. Heilongjiang Province | Daxinganling. A fire fox appears in the snow.
Audio
Keyframes
Video Clips
Comments
- So beautiful! 0
- Hope people don't hurt them. 1
- Fake! The saturation of this video has been adjusted! 10
User
Not verified.
I'm from ..., interested in ...
BERT
VGGish
VGG19
C3D
BERT
BERT
text
Cross-Modal Transformer
audio
text
Cross-Modal Transformer
text
vis
Σ
Transformer
Fake
Real

Training:

- Pretrained models (BERT, VGGish) for feature extraction.
- Fivefold cross-validation with event-level splits.
- Loss: Binary cross-entropy.

Results

We trained on 1000 videos, including 517 fake news videos and 483 real news videos.
The results metrics (Acc, F1, Precision, Recall) compared with those in the paper are as follows in the table.

Metric	TF Re-implement	Original Pytorch SVFEND
Accuracy	72.50% ± 8.13%	79.31% ± 2.75%
F1-Score	63.79% ± 14.44%	79.24% ± 2.79%
Precision	82.60% ± 2.40%	79.62% ± 2.60%
Recall	66.52% ± 10.28%	79.31% ± 2.75%

Challenges

Challenges:

- The FakeSV dataset is large and rich in multimodal information (videos, audios, comments, profiles).
- Some modules originally designed in PyTorch (e.g., Vggish) do not perform consistently when reimplemented in TensorFlow, leading to compatibility problems and unstable results.

Future Work:

- Adding more transformer layers and increasing model capacity to better capture complex multimodal interactions between text, audio, and visuals.
- Conduct ablation experiments to systematically evaluate the contribution of each modality (title, transcript, audio, visual frames, social context) to overall performance, guiding further model refinements.
- Implement data caching, memory mapping, and efficient batching strategies to speed up dataset reading and reduce memory usage during training.

Reflection

We successfully built a complete, working version of our system and gained valuable, data-backed insights into how combining text, audio, and visual modalities improves fake news detection in short videos. While we didn’t fully achieve our most ambitious performance targets, we are proud of the codebase we developed and the hands-on experience we gained working with multimodal models.

Although our model did not reach the performance levels reported in the original paper, analyzing the evaluation metrics was informative. We observed lower accuracy, F1 score, and recall—but higher precision—highlighting the challenges posed by limited data and an imbalanced dataset skewed toward real news samples. Given that we only utilized a subset of the dataset, we believe our results remain reasonable and indicative of the model's potential.

As the project progressed, we identified several components—particularly the co-attention transformers and multimodal fusion layers—that required deeper integration with TensorFlow's Keras API. Unlike PyTorch's dynamic computation graph, TensorFlow necessitated a more tailored approach. Consequently, we moved beyond a simple conversion and instead reimplemented these components using TensorFlow-native features such as `tf.keras.layers.MultiHeadAttention` and custom training loops.

A major pivot occurred in our feature extraction pipeline. Initially, we attempted to replicate pre-trained models like VGGish and C3D in TensorFlow. However, compatibility issues and the limited availability of TensorFlow-native pre-trained weights—especially for C3D—led us to replace or reimplement key parts of the pipeline. In hindsight, dedicating more time upfront to exploring the TensorFlow ecosystem for equivalent tools and models would have significantly reduced the effort spent during this phase.

To improve training efficiency, we propose a three-layer caching pipeline that offloads deterministic, computationally expensive steps from the training loop:

1. **Offline feature caching**: Run BERT, VGG19, C3D, and VGGish once per video, then store the resulting tensors in sharded LMDB or Parquet files on local NVMe storage.

2. **Sample caching**: During training, load these precomputed tensors to assemble "ready-to-train" samples, storing them in RAM via `tf.data.cache()`, Torch's cache, or memory-mapped NumPy arrays.

3. **Elimination of Python-side preprocessing**: This setup removes the need for runtime padding and collation, resulting in faster training and less overhead.

Another key takeaway from this project was the importance of structured team collaboration. With everyone balancing classes and internship or job applications, our weekly stand-ups and shared task tracker proved essential. The tracker kept everyone aligned, helped us schedule meetings efficiently, and ensured that no one fell behind. Most importantly, we learned how to coordinate effectively as a team under tight deadlines.