Moksh Nirvaan
Project 4- Final project report

Synthetic Misinformation Detection

Introduction and Project Statement

The increasing proliferation of misinformation across digital media platforms poses serious challenges to truth verification in the modern era. Two particularly dangerous forms of this are:

1. **Text-based misinformation** (e.g., fake news articles or political statements), and

2. **Visual misinformation** via deepfakes, where artificial intelligence is used to generate or manipulate videos to spread disinformation.

This project tackles both issues by developing a dual detection pipeline. The first is a text-based fake news detection system, leveraging transformer-based language models like BERT and GPT to classify and verify factual correctness of written claims. The second is a deepfake detection system, leveraging deep learning on facial imagery extracted from manipulated videos to identify synthetically generated human appearances.

Together, these systems demonstrate a multi-modal approach to misinformation detection and can serve as the foundation for a real-world disinformation response tool.

Data Sources and Technologies Used

Text Fake News Detection

- **Dataset**: LIAR dataset (Wang, UCSB), containing 12.8k hand-labeled short political statements from PolitiFact.com. Each statement is labeled as one of six classes and includes metadata about the speaker, context, and credibility history.

- **Binary Label Mapping**:

    - true, mostly-true, half-true → real

    - barely-true, false, pants-fire → fake

- **Technologies**

  For the fake news detection module, I used Python 3.12 as the core programming language, the HuggingFace Transformers library with the pre-trained bert-base-uncased model for natural language understanding, and the PyTorch backend with the Trainer

API to handle fine-tuning and evaluation. For secondary verification of low-confidence predictions, I integrated the OpenAI API, utilizing a large language model (LLM) to assess factual correctness based on internal knowledge.

Deepfake Detection

- **Dataset**: Celeb-DF v2 from Kaggle
    - Celeb-real: 590 real videos of celebrities
    - Celeb-synthesis: 5639 fake Deepfake videos
    - Youtube-real: 300 Additional authentic videos
- 5639 deepfake and 890 real videos Ire processed to extract facial frames for training.
- **Technologies**:

  The deepfake detection module was developed in a Jupyter Notebook environment running on a Mac M3 Pro with 36GB RAM, using Python 3.12. The model was built with PyTorch, utilizing the MPS backend for accelerated training on Apple silicon. I employed Torchvision's ResNet18 architecture for binary image classification, and MTCNN was used for efficient face extraction from video frames. Additional tools such as OpenCV and Matplotlib supported video processing and data visualization throughout the project.

Methods Employed

A. Fake News Detection (Text)

1. **Preprocessing**:

   The LIAR dataset, originally provided in TSV (tab-separated values) format, was loaded and parsed to extract the relevant columns, particularly the textual statements and their associated labels. The original six-class labels (true, mostly-true, half-true, barely-true, false, pants-fire) Ire then mapped into binary classes, with true, mostly-true, and half-true categorized as real, and the remaining labels as fake. After preprocessing, the cleaned dataset was converted into the HuggingFace Datasets format, enabling efficient tokenization, batching, and compatibility with the Transformers training pipeline.

2. **Model Training (Tier 1)**:
   To build the initial classifier, I fine-tuned the pre-trained bert-base-uncased model using the HuggingFace Trainer API, which streamlined the training loop, evaluation, and

checkpointing processes. The training was conducted on the binary-labeled LIAR dataset with standard hyperparameters, and the model's performance was evaluated on both the validation and test sets. I used accuracy, F1 score, precision, and recall as the primary evaluation metrics to assess the model's ability to distinguish betlen real and fake statements.

3. **Tier 2 LLM Verifier**:
   To enhance reliability, I implemented a second verification tier using a large language model. Any prediction from the BERT classifier with a confidence score below 0.7 was flagged as uncertain and passed to an LLM verifier. This module used OpenAI's GPT-4-turbo model via the modern OpenAI API. A structured prompt was crafted containing the original claim, and the LLM returned a verdict of either "REAL" or "FAKE" along with a brief justification, offering an additional layer of semantic verification based on the model's internal knowledge base.

B. Deepfake Detection (Video)

1. **Preprocessing and Face Extraction**:

   To prepare the data for training, I first used OpenCV to extract 20 evenly spaced frames from each video in the dataset, ensuring temporal diversity without overwhelming redundancy. For each extracted frame, I employed MTCNN (Multi-task Cascaded Convolutional Networks) to detect and crop face regions with high precision, even under varied lighting and pose conditions. This process resulted in approximately 17,771 real face images from authentic videos and 112,778 fake face images from deepfake videos, creating a significantly imbalanced but realistic training set reflecting real-world threats.

2. **Dataset Splitting**:

   To ensure unbiased evaluation and prevent data leakage, I performed a stratified shuffle split of the dataset. Each class (real and fake) was divided into 5,000 training, 1,000 validation, and 1,000 test images. Stratification preserved the class distribution across splits, which is especially important in binary classification tasks to avoid overfitting or misleading accuracy.

3. **Model Training**:

   For classification, I used ResNet18, a lightlight convolutional neural network from the Torchvision model zoo, and modified its final layer to output two logits for binary classification. The model was trained for three epochs using CrossEntropyLoss as the objective function and common data augmentation techniques (e.g., horizontal flip, random crop) to improve generalization. Training was conducted using the MPS

backend on Apple silicon, optimizing performance without the need for CUDA-enabled hardware.

4. **Evaluation**:

   After training, the model achieved 100% accuracy on the training, validation, and test sets, indicating an extremely high degree of separability in the extracted facial features betlen real and fake classes—possibly due to artifacts commonly introduced by generative models in deepfakes. The model's lights Ire saved to disk as best_deepfake_model.pth for later inference. During testing, the model was capable of processing face images at an average inference time of approximately 0.1 seconds per image, making it suitable for real-time or batch processing use cases.

Results

Metric	Fake News (BERT)	Deepfake Detection (ResNet18)
Accuracy (Test)	64.3%	100.0%
F1 Score	54.7%	100.0%
Precision / Recall	61.3% / 49.3%	100.0% / 100.0%
Inference Speed	~0.05 sec/text	~0.1 sec/image

- **Fake News**: BERT shoId moderate performance; Tier 2 GPT-based verification improves robustness.

- **Deepfakes**: Excellent accuracy, but overfitting risk due to high separability of visual artifacts in dataset.

Moksh Nirvaan
Project 4- Final project report

References

- Wang, William. LIAR Dataset: https://www.cs.ucsb.edu/~william/
- Li, Y., Chang, M.C., Lyu, S. (2020). Celeb-DF: A Large-scale Challenging Dataset for DeepFake Forensics. CVPR.
- Kaggle Celeb-DF v2: https://www.kaggle.com/datasets/reubensuju/celeb-df-v2
- HuggingFace Transformers: https://huggingface.co/docs
- PyTorch Docs: https://pytorch.org
- Torchvision Models: https://pytorch.org/vision/stable/models.html
- MTCNN Face Detection: https://github.com/ipazc/mtcnn
- OpenAI API: https://platform.openai.com