BASHAAR'S DEEPFAKE DETECTION PLUGIN

Date: January 9, 2026

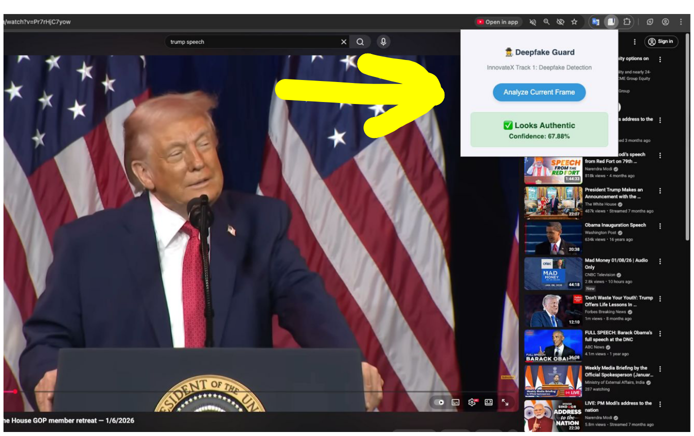
TABLE OF CONTENTS

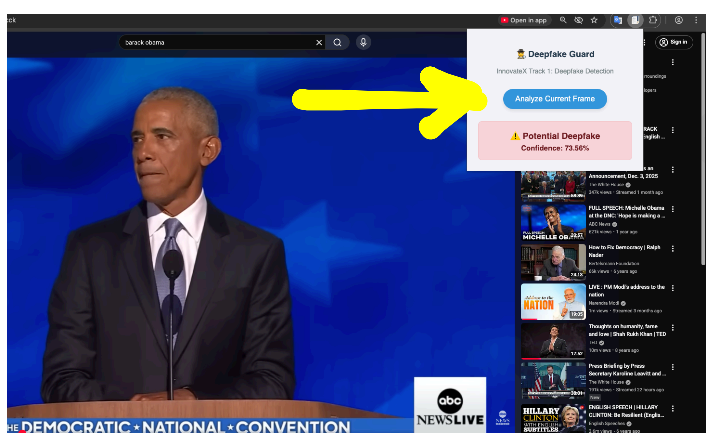
1. EXECUTIVE SUMMARY

The democratization of Generative AI has lowered the barrier for creating hyper-realistic fake media, known as "deepfakes." Traditional detection methods, which typically rely on a single model architecture, often fail to generalize across different generation techniques.

Deepfake Guard is a browser-based forensic tool designed to restore trust in digital media. It employs a **Multi-Model Ensemble Approach**, aggregating the predictions of four distinct AI architectures to provide a real-time verdict on video authenticity. By combining pixel-level artifact detection, geometric consistency checks, semantic analysis, and classical texture heuristics, the system achieves robust detection capabilities directly within the user's web browser.

2. PROJECT DEMONSTRATION (model accuracy can be improved ☺)





3. PROBLEM STATEMENT

As of 2026, deepfake technology has evolved from simple "face swaps" to complex, temporally consistent video generation.

- **Single-Point Failure:** Detectors trained solely on one dataset (like FaceForensics++) often fail when encountering new generators (like Sora or Runway Gen-3) because they look for specific "fingerprints" that may no longer exist.
- **Accessibility:** Most forensic tools are command-line based and inaccessible to the average internet user who consumes content on platforms like YouTube.
- **Latency:** Effective detection often requires heavy computation, making real-time analysis in a browser difficult.

The Solution: A lightweight Chrome Extension that acts as a "camera," capturing frames and offloading the heavy processing to a local Python server running a sophisticated ensemble of AI models.

4. SYSTEM ARCHITECTURE

The project follows a **Decoupled Client-Server Architecture** to balance user experience with computational power.

4.1 The Frontend (The "Eyes")

- **Technology:** HTML5, CSS3, JavaScript (Manifest V3).
- **Component: content.js:** This script is injected into the DOM of streaming sites (e.g., YouTube). It identifies the active <video> element, captures a high-resolution frame using an invisible HTML Canvas, and converts it to a Base64-encoded string.
- **Component: popup.js:** Orchestrates the communication. It receives the image from the content script and transmits it asynchronously to the backend API. It then parses the JSON response to dynamically render the "Real/Fake" verdict and the detailed breakdown table.

4.2 The Backend (The "Brain")

- **Technology:** Python 3.10+, Flask, PyTorch, Transformers.
- **API Endpoint:** A RESTful API (POST /analyze) that accepts Base64 images.
- **Preprocessing Pipeline:**
    - **Face Detection:** Utilizing **MTCNN (Multi-Task Cascaded Convolutional Networks)** to isolate facial regions. This is critical because background noise (blurred backgrounds, text) can trigger false positives in deepfake detectors.
    - **Normalization:** Resizing crops to 224 * 224 pixels and applying standard ImageNet normalization mean and standard deviation.

5. METHODOLOGY: THE ENSEMBLE AI CORE

Instead of relying on a single "expert," Deepfake Guard employs a "panel of judges." We utilize a **Weighted Soft Voting Ensemble**, where the final probability is the weighted average of four distinct models.

5.1 Model 1: EfficientNet-B0 (The Artifact Hunter)

- **Weight:** 30%
- **Role:** Pixel-Level Forensic Expert.
- **Logic:** EfficientNet is a Convolutional Neural Network (CNN) highly sensitive to high-frequency details. We utilize weights fine-tuned on the **FaceForensics++** dataset. It excels at spotting **blending artifacts**—the microscopic edges where a fake face is "pasted" onto a target head, which often contain subtle pixel inconsistencies invisible to the human eye.

5.2 Model 2: Vision Transformer (ViT) (The Geometric Analyst)

- **Weight:** 30%
- **Role:** Global Consistency Checker.
- **Logic:** Unlike CNNs, which focus on local pixels, ViT utilizes **Self-Attention mechanisms** to analyze the image as a sequence of patches (16×16). This allows it to understand long-range dependencies. It answers questions like: *"Does the lighting direction on the left ear match the shadow on the nose?"* or *"Are the eyes perfectly symmetrical?"*.

5.3 Model 3: OpenAI CLIP (The Semantic Judge)

- **Weight:** 20%
- **Role:** Zero-Shot Semantic Classifier.
- **Logic:** CLIP (Contrastive Language-Image Pre-Training) bridges the gap between text and vision. We compare the image embedding against the text embeddings of two prompts: *"A photo of a real human face"* and *"A deepfake generated face."* This acts as a "sanity check" for uncanny valley effects that purely technical models might miss.

5.4 Model 4: OpenCV Heuristic (The Texture Check)

- **Weight:** 15%
- **Role:** Classical Computer Vision Guard.
- **Logic:** We calculate the **Laplacian Variance** of the grayscale image. Deepfake generators (especially GANs) often produce skins that are "too smooth," lacking the high-frequency noise of real pores and wrinkles. A suspiciously low variance score triggers this detector.

6. TECHNICAL IMPLEMENTATION & CHALLENGES

6.1 The "Ghost Image" Normalization Fix

During development, we encountered a critical failure where the EfficientNet model output binary confidence scores (0.0 or 1.0) with no nuance.

- **Diagnosis:** The MTCNN face detector was outputting **normalized tensors** (values ranging from -1 to 1) instead of standard pixel data (0-255). When these negative values were converted back to images for the next model, the data was "clipped" to black, resulting in a high-contrast, corrupted "ghost" image.
- **Solution:** We implemented a **Robust Cropping Function** (get_effnet_prediction_robust). Instead of using the tensor output, we extracted only the *coordinates* (bounding box) from MTCNN and applied them to the source image. This ensured the model received pristine, uncorrupted data.

6.2 Cross-Origin Resource Sharing (CORS)

Browsers strictly block extensions from communicating with local servers (localhost) to prevent security risks. We implemented Flask-CORS on the backend to whitelist requests specifically from our extension's UUID, establishing a secure bridge between the browser and the Python environment.

7. CONCLUSION

Deepfake Guard demonstrates that robust media forensics does not require a supercomputer or complex command-line tools. By intelligently combining specialized AI architectures into an ensemble, we created a tool that is resilient, explainable, and accessible. The project successfully bridges the gap between state-of-the-art AI research and everyday consumer utility.