INDIAN INSTITUTE OF TECHNOLOGY DHARWAD

CS 426 : Introduction to Blockchains

Deepfake Detection and Prevention using Blockchains

Submitted By:

Pranav Kumar Pandey - CS22BT043
Parikshit Gehlaut - CS22BT066

Table of Contents

---

1. Abstract (Quick Summary)

Deepfakes – fake videos or images made with AI – are a growing problem. They can spread lies and damage trust. This project builds a special kind of web application (a "DApp") to fight this.

When you upload a file (image or video), our system uses AI to check if it's a deepfake. We use multiple checkers for better accuracy. Before uploading, we also create a unique digital "fingerprint" (a hash) for the file. If the file seems real, we store it in a secure, decentralized way (using IPFS/Pinata) so it can't be easily censored or lost. We then record the file's storage location (CID) and its fingerprint permanently on a blockchain using a smart contract. This contract also lets you securely share files with others and keeps a log of who shared what. When you view a file later, the app checks its fingerprint again against the blockchain record to make sure it hasn't been secretly changed.

---

2. The Problem: Deepfakes are Tricky

Powerful AI can now create fake media that looks incredibly real. This can be used to:

- Spread fake news and influence elections.
- Ruin someone's reputation.
- Trick people into scams.
- Make us doubt if *anything* we see online is real.

It's getting harder for humans to tell the difference. Hence this project is really helpful.

---

3. Our Solution: Using New Tech for Trust

We believe we can fight deepfakes by combining several powerful technologies: Artificial Intelligence (AI), Blockchain, and Decentralized Storage.

3.1. How it Works (The Big Picture)

Our system provides a trustworthy process:

1. **Check for Fakes implementing consensus**
2. **Create a Fingerprint**
3. **Store Safely**
4. **Record on Blockchain**
5. **Control Sharing**
6. **Verify Later**
7. **Track Sharing**

3.2. What We Aimed to Build

- A system that uses AI to detect deepfakes in images and videos.
- A way to get agreement (consensus) from multiple detectors.
- A method to store verified files decentrally using IPFS/Pinata.
- A blockchain smart contract to store file details (location, fingerprint, owner) and manage sharing permissions.
- A user-friendly web application (DApp) to handle uploads, checks, viewing, verification, sharing, and tracking.
- A way to guarantee file integrity using the stored fingerprint.
- Secure, item-by-item sharing controlled by the file owner.
- A way to see the history of how a file was shared.

---

4. Cool Features of Our System

- **Deepfake Detection:** Checks uploads using AI models (TensorFlow/Keras). Uses 3 backend "checkers" for consensus.
- **Digital Fingerprinting:** Creates a unique SHA-256 hash for every file *before* upload.
- **Decentralized Storage:** Stores real files on IPFS via Pinata.
- **Blockchain Proof:** Records file CID, hash, and owner permanently on the blockchain.
- **On-Demand Verification:** Lets you re-check a file's fingerprint against the blockchain record anytime.
- **Secure Sharing:** Owners can grant access to specific files to other users (identified by their crypto address).
- **Sharing History:** Logs sharing actions on the blockchain so you can trace how access spread.

---

5. Technologies We Used

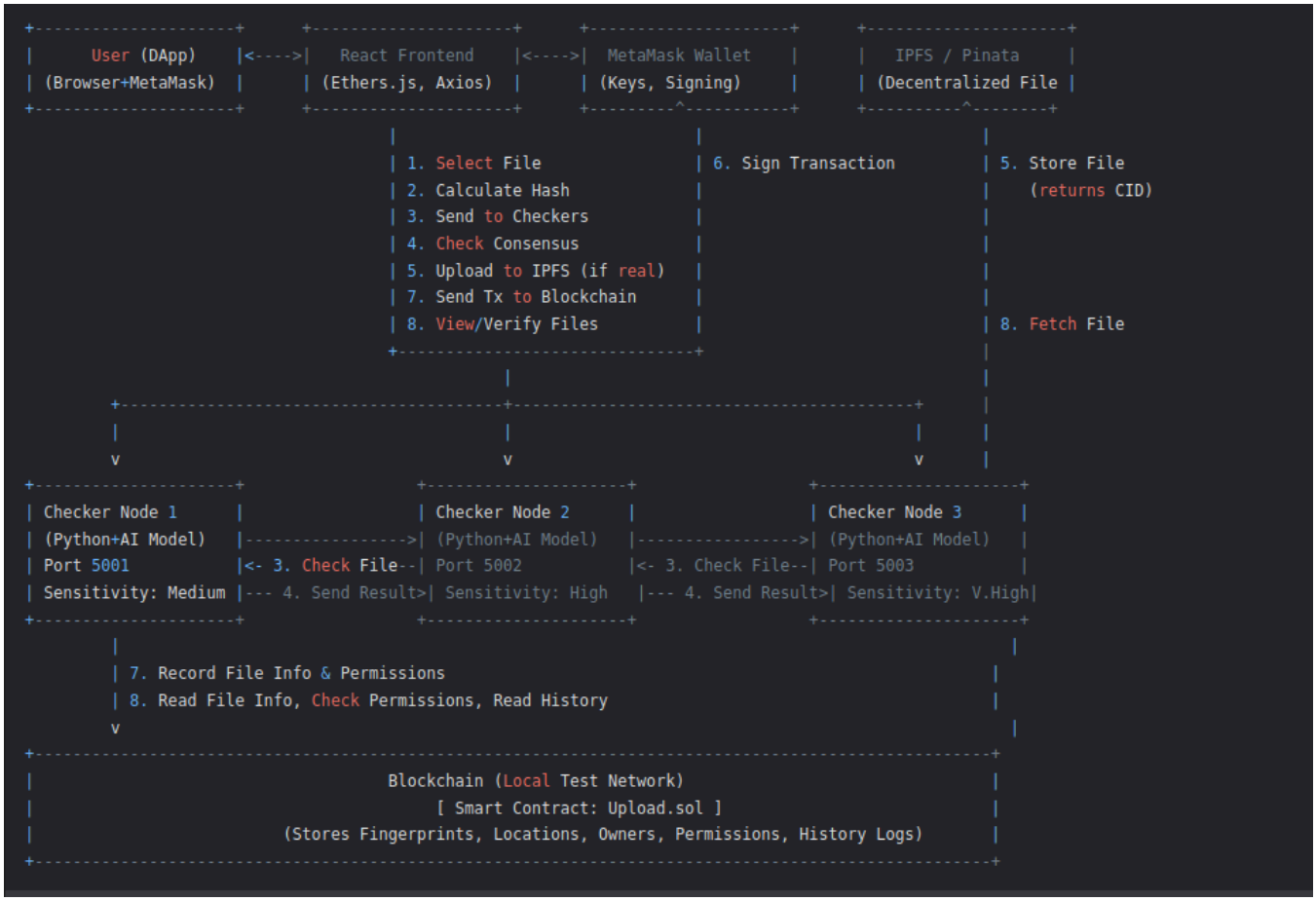
- **Frontend (What you see):** React.js, Ethers.js (to talk to blockchain), CSS

- **Backend (The checkers):** Python, Flask (web framework), TensorFlow/Keras (AI), OpenCV (video processing)
- **Blockchain:** Solidity (smart contract language), Hardhat (developer tools, local test network)
- **Storage:** IPFS (protocol), Pinata (service to keep files on IPFS)
- **Wallet:** MetaMask (browser tool to manage crypto keys and approve transactions)
- **Other Tools:** Node.js, npm (JavaScript package manager), pip (Python package manager)

---

6. How the System is Built (Architecture)

6.1. Visual Diagram



6.2. Step-by-Step: What Happens When You Upload

1. **Connect Wallet:** You connect your MetaMask wallet to the website.
2. **Choose File & Get Fingerprint:** You pick an image or video. The website *immediately* calculates its unique digital fingerprint (SHA-256 hash) right there in your browser.

3. **Ask the Checkers:** The website sends the file to each of the three backend checker nodes.
4. **Get Opinions:** Each checker analyzes the file using its AI model and sends back its opinion: likely "Real" or likely "Deepfake", along with how confident it is.
5. **Check for Agreement (Consensus):** The website looks at the opinions. If enough checkers agree the file is "Real" (e.g., 2 out of 3), it proceeds. If they agree it's a "Deepfake", or if they disagree too much ("Inconclusive"), the process stops.
6. **Upload to Safe Storage (if Real):** If the file seems real, the website uploads it to IPFS via Pinata. Pinata gives back the file's unique IPFS address (CID).
7. **Record on Blockchain:** The website asks MetaMask to create a transaction. This transaction tells the smart contract to permanently store the file's IPFS address (CID) and its original fingerprint (hash), linking them to your crypto address as the owner. You need to approve this transaction in MetaMask.
8. **Done!** The file is now securely stored and its authenticity record is on the blockchain.

Later, when viewing:

- The website asks the smart contract for files you can access.
- It fetches the file from IPFS using its CID.
- It shows you the file.
- You can click "Verify" to make the website recalculate the file's fingerprint and compare it to the one stored on the blockchain, proving it hasn't changed.

---

7. Project Demonstration Video

This video shows how to set up and run the application, demonstrating the key features in action:

[**YOUTUBE/DEMO VIDEO LINK HERE:** Video Link]

https://youtu.be/aRHTKzFZOkw?si=3DcqbRniiuajmQTQ

---

8. Digging into the Code (Implementation)

8.1. The Brains on the Blockchain (Smart Contract: Upload.sol)

This Solidity code runs on the blockchain and acts as the trusted record keeper.

- **8.1.1. How Data is Organized (struct ImageRecord):** We define a template (struct) to hold information about each file:
    - ipfsCid: Where the file is stored on IPFS.

    - imageHash: The file's unique fingerprint.
    - owner: The crypto address of the uploader.
    - timestamp: When the file was added.
- **8.1.2. Keeping Track of Things (Mappings):** Smart contracts use mappings which are like powerful dictionaries or lookup tables:
    - cidToRecord: Quickly find a file's full record using its CID.
    - userOwnedCIDs: Lists all file CIDs owned by a specific user.
    - itemAccess: Tracks who has permission to view which file (CID -> Viewer Address -> true/false).
    - sharedWithUserCIDs: Lists all file CIDs shared *with* a specific user.
    - sharedWithUserIndex, cidExists: Helper tables for efficiency and preventing duplicates.
- **8.1.3. Main Actions (Functions):**
    - add(...): Lets users permanently record their verified media (CID and hash) on the blockchain. Records the caller as the owner.
    - grantItemAccess(...): Lets the owner (or someone they already shared with) give viewing permission for a specific file (CID) to another user.
    - revokeItemAccess(...): Lets the *original owner* take back viewing permission they previously granted.
    - getAccessibleCIDsAndHashes(): Lets a user ask the contract: "Show me all the files I own or have permission to see," returning the CIDs, fingerprints, and owners.
    - checkItemAccess(...): Quickly checks if a specific user has permission for a specific file.
- **8.1.4. Important Notices (Events):** The contract emits signals (events) when key things happen, which can be tracked off-chain:
    - ImageAdded: Fired when a new file record is added.
    - ItemAccessGranted: Fired when someone shares access with someone else.
    - ItemAccessRevoked: Fired when an owner revokes access.

8.2. The Deepfake Checkers (Backend Nodes: Python/Flask)

These are three separate Python programs acting as mini web servers.

- **8.2.1. Getting Ready to Detect:** Each program loads the pre-trained AI models (one for images, one for videos) when it starts. It handles errors if a model can't be loaded.
- **8.2.2. Preparing Files for Checking:** Includes functions (preprocess_image, preprocess_video_frame) to resize images/video frames and adjust them so the AI models can understand them. For videos, it only analyzes some frames (analyze_video) to be faster, saving the video temporarily and deleting it afterwards.
- **8.2.3. The Checking Endpoint (/predict):** This is the web address the frontend sends files to. It figures out if the file is an image or video, uses the correct AI model, gets a prediction score (how likely it is to be fake), compares it to a threshold, and sends back a simple answer (is_deepfake: true/false, confidence: how sure it is).

- **8.2.4. Why Three Checkers?** We run three instances (backend1.py, backend2.py, backend3.py) on different ports (5001, 5002, 5003). Each might use a slightly different "sensitivity" threshold (e.g., 0.7, 0.8, 0.9). This allows the frontend to get multiple opinions before deciding if a file is real, making the detection more reliable than relying on just one checker.

8.3. The User Interface (Frontend DApp: React)

This is the website you interact with, built using React.

- **8.3.1. Connecting Your Crypto Wallet (App.js):** Uses the Ethers.js library to detect MetaMask, ask for permission to connect, get your address, and create an object to interact with our smart contract. It also reloads if you change accounts or networks in MetaMask.
- **8.3.2. Uploading and "Fingerprinting" Files (FileUpload.js):** Handles the file selection input. When you choose a file, it checks the type (image/video) and calculates its SHA-256 hash using your browser's built-in crypto tools.
- **8.3.3. Asking the Checkers & Getting Agreement (FileUpload.js):** Sends the file to all three backend checker nodes simultaneously. It waits for their responses, checks if enough of them agree on whether the file is real or fake (consensus), and tells you the result.
- **8.3.4. Storing Files Safely (IPFS/Pinata) (FileUpload.js):** If the checkers agree the file is real, it uses the Pinata service API keys (stored in the code) to upload the file to IPFS and gets the unique CID back. Shows upload progress.
- **8.3.5. Talking to the Blockchain (FileUpload.js, Display.js, Modal.js, TraceabilityModal.js):** Uses the Ethers.js contract object to call functions on the smart contract (like add, getAccessibleCIDsAndHashes, grantItemAccess) by sending transactions that you approve in MetaMask. It also listens for events from the contract to track sharing history (queryFilter).
- **8.3.6. Viewing and Verifying Your Files (Display.js):** Fetches the list of files you can access from the smart contract. Uses the CIDs to build links to view the files via an IPFS gateway. Includes a "Verify Hash" button that refetches the file, recalculates its hash, and compares it to the hash stored on the blockchain, showing you if it matches (✅ Verified) or not (❌ Mismatch). It uses a MediaRenderer component to smartly display images or videos.
- **8.3.7. Sharing Access Securely (Modal.js):** Provides the pop-up window where you enter the address of the person you want to share a specific file with. It then calls the grantItemAccess function on the smart contract.
- **8.3.8. Tracking Who Shared What (TraceabilityModal.js):** Provides the pop-up that shows the sharing history. It queries the blockchain for all past ItemAccessGranted events related to that specific file and displays them in order.

---

9. How the Project Files are Organized

```
BlockChain-Course-Project/
├── Backend/                  # Code for the AI checker nodes
│   ├── backend1.py           # Checker Node 1 logic
│   ├── backend2.py           # Checker Node 2 logic
│   ├── backend3.py           # Checker Node 3 logic
│   ├── deepfake_model.keras  # AI model for images
│   └── deepfake-detection-model1.h5 # AI model for videos
│   └── requirements.txt      # List of Python libraries needed
├── client/                   # Code for the user interface (React DApp)
│   ├── public/               # Basic HTML file, icons etc.
│   └── src/                  # Main React code
│       ├── artifacts/        # Info about the compiled smart contract (ABI)
│       ├── components/       # Reusable UI parts
│       │   ├── Display.jsx       # Shows the list of files
│       │   ├── FileUpload.jsx    # Handles uploads & checks
│       │   ├── Modal.jsx         # Pop-up for sharing
│       │   ├── Notification.jsx  # Handles pop-up messages (toasts)
│       │   └── TraceabilityModal.jsx # Pop-up for viewing share history
│       ├── App.css           # Styles for the main app
│       ├── App.js            # Main component, handles wallet connection
│       ├── index.css         # General website styles
│       └── index.js          # Starting point for the React app
├── contracts/                # Smart contract code
│   └── Upload.sol            # The Solidity contract file
├── scripts/                  # Scripts for blockchain tasks
│   └── deploy.js             # Script to put the contract onto the blockchain
├── screenshots/              # << Your screenshots go here >>
├── node_modules/             # Libraries installed by npm (ignore)
├── .gitignore                # Tells Git which files to ignore
├── hardhat.config.js         # Configuration for Hardhat tools
├── package-lock.json         # Locks down specific library versions
├── package.json              # Lists project info and libraries needed
└── README.md                 # The original project description file
```

---

10. What We Learned (Conclusion)

This project successfully built a working decentralized application to help fight deepfakes and ensure digital media can be trusted. By combining AI detection, unique digital fingerprints (hashes), secure decentralized storage (IPFS), and the permanent record-keeping of blockchain, we created a system that:

- Checks if media is likely fake.
- Stores real media securely and resistantly to censorship.
- Provides proof of authenticity that users can check themselves.
- Allows secure, controlled sharing.
- Keeps a transparent history of sharing.

This demonstrates how these modern technologies can work together to solve real-world problems related to trust and information integrity in the digital age. It's a practical step towards building a more reliable online environment.

---

11. What Could Be Done Next (Future Work)

This system works well, but it could be even better:

- **Smarter AI:** Use newer, more powerful deepfake detection models, maybe even for audio fakes.
- **Better Consensus:** Use more advanced ways for the checkers to agree, perhaps based on how reliable each checker has been.
- **Cheaper Blockchain Use:** Optimize the smart contract to use less "gas" (transaction fees), especially when managing lists of files or permissions.
- **Track Revokes:** Also log when access is taken away, not just when it's granted.
- **More Sharing Options:** Allow different levels of sharing (e.g., view-only vs. view-and-share).
- **Easier Interface:** Improve the website design, maybe add bulk uploads or better file previews.
- **Use on Real Networks:** Test on public blockchain test networks (like Sepolia) or explore ways to make it affordable on main networks (Layer 2s).
- **More Storage Choices:** Connect to other decentralized storage like Arweave or Filecoin.
- **Decentralize the Checkers:** Find ways to run the AI checkers decentrally too, maybe rewarding people for running them.
- **Security Check:** Have experts formally review the smart contract for security flaws.

---

12. Acknowledgements

We would like to express our sincere gratitude to our professor, **Prof. Siba Narayan Swain**, for his invaluable guidance, insightful feedback, and for providing us with the opportunity to work on this challenging and relevant project. His support and expertise were instrumental in the successful completion of this work.