White Paper: Decentralized Originality Verification: A Secure AI-Blockchain Approach to Content Posting in Social Platforms

Executive Summary

Konthokosh is a proposed decentralized platform for sharing creative text (stories, poems, etc.) with built-in plagiarism prevention. It combines **AI-based content analysis** and **blockchain notarization** to ensure originality. When a user submits a post, the system generates a semantic embedding of the text, searches for similar content in its database (using a *retrieval-augmented* AI pipeline), and computes a similarity score. If the post is unique (below a plagiarism threshold), it is **recorded on-chain**: the content's hash, author's wallet address, timestamp, and any related references are stored in a smart contract. This provides an immutable, timestamped proof of authorship. By contrast to centralized checks, our approach leverages blockchain's trustless architecture so "no blockchain-based notarization tool [currently] integrates anti-plagiarism" detection. Embedding documents in a vector index and running exact similarity metrics (e.g. BERT-based embeddings) helps catch paraphrased or translated copies. The blockchain ledger then **decentralizes trust** – no single authority is needed to validate originality – and maintains an immutable evidence trail.

Introduction & Motivation

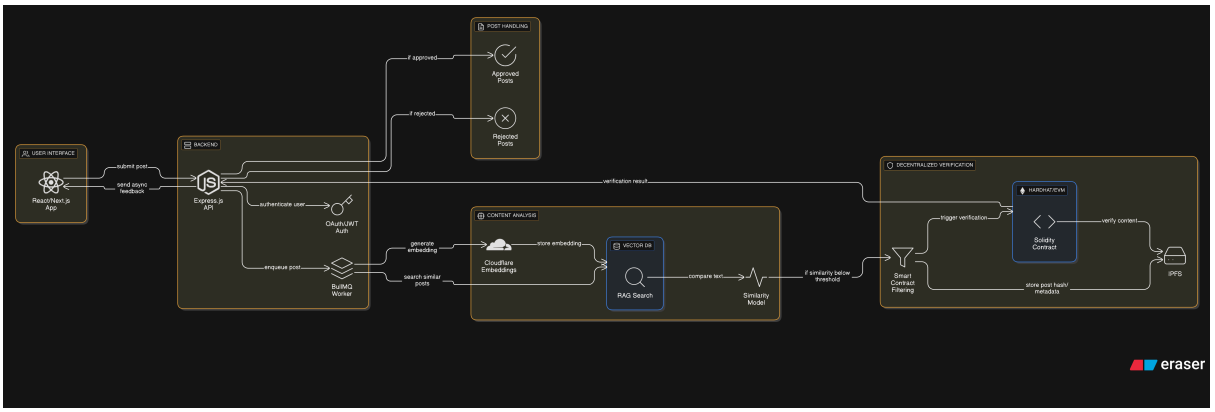
Online social platforms make it easy to share content, but also make plagiarism a widespread risk. Traditional plagiarism checks rely on centralized authorities or proprietary databases, and cannot prevent dishonest actors from copying others' posts. We address this by decentralizing trust: **blockchain's immutability** provides a tamper-proof timestamped record of every submission, while **AI-based similarity analysis** identifies any suspicious overlaps. As Sarcinella *et al.* point out, blockchain can eliminate the need for a central notary: "trustless" transactions and timestamping let parties verify authenticity without a third party. In particular, our system functions much like a decentralized notary for text: when content is submitted, its hash (e.g. SHA256) is stored on-chain, proving it existed at a given time. Any later similar content can be immediately compared against this immutable record. This paradigm is similar to how digital music platforms use blockchain for copyright proof: storing a *content hash on-chain serves as proof of existence*, coupled with hash-based uniqueness

checks. Likewise, if a plagiarist appears, the chain's earlier record "can serve as evidence in defense of the original owner's rights". In short, Konthokosh leverages blockchain to *notarize* user-submitted content and AI to *verify* it, creating a secure, decentralized originality-verified posting system.

System Architecture Overview

The platform has three main layers: **Frontend (UI and auth)**, **Backend (AI analysis and storage)**, and **Blockchain (ledger)**. The high-level flow is:

```
User (Web Browser)
   ── MetaMask login via Clerk ──▶ Frontend (React + Clerk)
   ── Submit new post ──▶ Backend Server (RAG, Vector DB, Plagiarism E
ngine)
   ──▶ Blockchain (Smart Contracts)
```



- **User Authentication:** Users sign in with their MetaMask Ethereum wallet (via Clerk.dev integration). MetaMask provides a unique wallet address as the user's ID, ensuring each post is linked to a verifiable identity. Clerk can optionally collect readable info (email/username) alongside the wallet if needed, but the on-chain identity is the wallet address.
- **Content Submission:** An authenticated user submits a story or poem through the web UI. The frontend bundles the content with the user's session token and wallet ID and sends it to the backend via a REST/API call.
- **Embedding & Storage:** On the backend, we generate a semantic embedding of the text (e.g. using a transformer model like BERT or a sentence-embedding LLM). This vector and the original text are saved in our database (a document store or SQL DB). The vector

is also added to a vector-index/search engine (e.g. Pinecone, Faiss) for fast similarity queries.

- **Similarity Retrieval (RAG):** The new post's embedding is queried against the stored index to retrieve the nearest-neighbor documents. This *retrieval step* (akin to Retrieval-Augmented Generation) brings in context: we fetch the top-N most semantically similar existing posts.
- **Plagiarism Check:** The backend then applies precise similarity metrics on the text of each candidate. For example, we can use cosine similarity or string-distance on TF-IDF or embedding vectors. If *any* retrieved document exceeds a set similarity threshold (e.g. 80%), the new post is flagged as potential plagiarism. This two-phase approach (vector retrieval + text metric) can detect even paraphrased or translated duplicates. If the post is flagged, the system can notify the user or moderator (implementation choice). If it's under the threshold, it is considered original.
- **Blockchain Registration:** For original posts, the backend computes a content-hash (e.g. SHA256 of the text or IPFS CID) and calls a smart contract function (via the user's MetaMask) to record it. The contract stores:
  - The *hash* (fingerprint) of the content,
  - The author's address (msg.sender, i.e. their wallet),
  - A timestamp (block time),
  - Any reference IDs of similar content (if we choose to record "near duplicates").

  The contract then assigns a unique content ID (or mints an ERC-721 token) for this entry. The on-chain ledger now contains an immutable notarization of the post.
- **Immutable Proof:** Once on-chain, the content entry is permanent. The transaction hash or contract event becomes the *content's blockchain ID*. This ID can be returned to the user as proof of originality. In case of any future dispute or duplicate posting, the chain's record provides cryptographic evidence: as noted in digital copyright systems, even if blockchain can't "prevent" plagiarism, it "offers proof of ownership at a specific time" to back up legal claims. In effect, Konthokosh becomes a distributed notary: every post is time-stamped and verifiable without trusting a central authority.

User Interface and Identity

- **MetaMask Login (Web3 Auth):** The frontend is a modern web app (e.g. React) that integrates MetaMask via Clerk's Web3 auth. When users click "Sign In", they approve a

wallet signature, and Clerk returns a session. This uses the user's Ethereum address as their ID.

- **Front-End Flow:** The UI allows users to write or paste content, see previous submissions, and view originality reports. Upon submitting, the frontend sends the text and Clerk token to the backend. For blockchain registration, the frontend also triggers a MetaMask transaction (signed by the user) to the smart contract. The user's wallet thus directly signs the on-chain record of their own content.
- **User Experience:** The site (named "Konthokosh", Bengali for "Word Treasury") emphasizes easy content sharing and transparency. Users see the status of their posts ("Processing…", "Original", or "Duplicate"), and once a post is finalized, they can copy its blockchain content ID for proof. The design can display visual badges for verified originality.

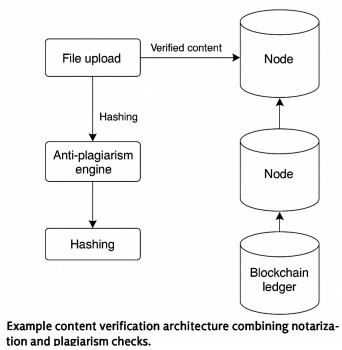
Content Analysis & Plagiarism Detection

- **Embedding Generation:** Every post is fed into a language model to produce a high-dimensional vector. We use a transformer model (e.g. BERT or RoBERTa) to capture contextual meaning. In tests, transformer embeddings significantly outperform simple keyword matching for detecting paraphrased content. For example, we can implement:

```
# (pseudocode for analysis)
embed = TransformerModel.embed(new_post_text)
vector_db.add(id=new_post_id, vector=embed, metadata={text, autho
r})
```

- **Vector Search (Retrieval):** The new post's vector is queried against the index. We use an ANN (approximate nearest neighbors) database (like Pinecone or an open-source index) to find the top K semantically nearest posts. This "retrieval" step is inspired by the RAG paradigm – we pull contextual candidates to "ground" our comparison. For example, Keita *et al.* outline a workflow that first vectorizes documents and builds an index, then checks new documents against it for plagiarism.
- **Text Similarity Metric:** For each candidate, we then compute a detailed similarity score. Options include cosine similarity of TF-IDF vectors, Jaccard similarity of n-grams, or even a second neural module. If *any* score exceeds a threshold (e.g. >80%), we flag it. This dual approach (semantic retrieval + quantitative matching) catches cases like paraphrased or translated plagiarism, which traditional tools miss.

- **Language Handling:** If multilingual support is needed, we can auto-detect language. If the content isn't in the primary language, we may translate it (e.g. via an API) to compare on a common basis, similar to techniques in academic plagiarism detection. The Pinecone workflow even includes a language-detector and translator before vectorizing.
- **Results:** The backend returns to the frontend either a "clear" pass or a warning with details (which posts were similar and what the similarity percentage was). If a post is flagged, the UI can alert the user and prevent on-chain registration.



Example content verification architecture combining notarization and plagiarism checks.

Figure: Transformer-based plagiarism detection pipeline. The new document (right) is vectorized and compared against stored vectors; similar candidates are then scored for exact text overlap. Green means "original", red means "duplicate". In this example workflow , an incoming text is preprocessed and embedded, then checked against the indexed corpus. High similarity triggers a plagiarism alert, while low similarity allows proceeding to blockchain recording.

Blockchain Integration & Smart Contracts

- **Blockchain Selection:** We can implement on a public or permissioned blockchain. MetaMask implies an Ethereum-compatible chain. For higher throughput and lower fees, a Layer-2 (e.g. Polygon, Arbitrum) or a private consortium chain could be used. Regardless, the core logic is in smart contracts (Solidity on EVM, for instance).
- **Contract Data Model:** We define a contract (e.g., `ContentRegistry`) with a struct:

```
struct Content {
    address author;
    bytes32 contentHash;
    uint256 timestamp;
    uint256[] similarIds;
}
mapping(uint256 ⇒ Content) public contents;
uint256 public nextId;
```

When a new original post is accepted, the contract function `registerContent(bytes32 hash, uint256[] refs)` is called by the author (msg.sender). It does:

- Check that `contents[nextId]` is empty (to avoid re-registration of same ID).
- Save `contents[nextId] = Content(msg.sender, hash, block.timestamp, refs)`.
- Emit an event `ContentRegistered(nextId, msg.sender, hash, refs)`.
- Return `nextId++` as the content's blockchain ID.

- **Proof of Originality:** By storing `hash` (e.g. SHA256 of text or an IPFS content ID) on-chain, we don't expose full content, but we guarantee integrity. Only the hash is visible, preserving privacy. The blockchain record ties this hash to the user's wallet and a timestamp. As in music copyright systems, this on-chain "proof-of-existence" means later duplicates cannot override the original author.
- **Handling Similar Posts:** If a later post is flagged similar to an existing one, the contract call can include `refs` linking the old content ID(s). This way, the ledger explicitly records that content A and B are related by similarity. For example, if Post 2 is 90% similar to Post 1, registering Post 2 on-chain might pass `refs = [1]`. Downstream applications can then detect plagiarism links by reading these references.
- **NFT Option:** Alternatively, each post could mint an ERC-721 token representing that content. The NFT's metadata might include the hash and similarity data. This turns content ownership into a tradable asset. (Sarcinella *et al.* suggest generating NFTs after verification, though our core system just needs the ID link.)
- **Immutability & Evidence:** Once on-chain, records can't be changed. In disputes, one can present the blockchain entry as evidence. As noted in prior work: "the system's immutable records can serve as evidence in defense of the original owner's rights". If a plagiarized copy were posted later, the chain immediately shows the original was earlier. This dissuades plagiarism and can support legal claims.
- **Decentralization:** Importantly, the blockchain layer ensures there's no single point of failure. Even if our website or backend went down, the content registry lives on-chain, replicated by all nodes. Sarcinella *et al.* emphasize that blockchain eliminates reliance on a trusted middleman for notarization. We fully leverage that property: originality verification is not done by a centralized server, but by consensus among nodes.

Figure: Example content verification architecture combining notarization and plagiarism checks. In this conceptual diagram (adapted from Sarcinella *et al.*), an uploaded file is hashed (bottom left) and processed by an anti-plagiarism engine. Verified content is then broadcast to blockchain nodes, which write the transaction to the ledger. This illustrates how user

submissions can be concurrently **notarized** (via hash storage) and **checked for originality** before finalizing on-chain.

Storage Design (IPFS + Database)

We adopt a hybrid storage strategy for efficiency: critical proofs on-chain, full content off-chain. Following the music copyright model, our **storage layer** can combine: blockchain, IPFS, and a conventional database. Specifically:

- **Blockchain (Ledger):** Stores only essential, tamper-proof data (content hash, pointers, authorship). This is the most secure but also most costly layer, so we use it sparingly.
- **IPFS or Decentralized File System:** (Optional) If posts are large (e.g. long poems), the full text can be stored on IPFS. We would then put the IPFS CID (itself a hash) on-chain instead of raw text. IPFS "manages large files" and "provides distributed file storage with security", ensuring the content is retrievable even if our server is down. This maintains decentralization in both indexing (chain) and storage (IPFS).
- **Traditional Database (e.g. MySQL or Postgres):** We use a standard DB to store metadata (content ID, text, author, embed vector, similarity scores) for fast querying. This is just for performance; all crucial proofs remain in IPFS/blockchain. In effect, blockchain + IPFS + DB gives a balanced architecture: "Blockchain handles critical, tamper-proof data; IPFS manages large files; and MySQL stores metadata".
- **Security Balance:** The layered design optimizes for both security and efficiency. Only hashes and references go on-chain (secure), while rich queries on text are done in the database (efficient). This mirrors existing solutions for digital content, where "blockchain provides tamper-proof security, IPFS enables reliable storage of large files, and a traditional DB ensures efficient access to frequently used metadata".

System Workflow Example

1. **Login:** User opens Konthokosh web app and connects MetaMask. Clerk authenticates and returns the user's wallet address.
2. **Posting:** The user writes a story and clicks *Submit*. Frontend sends the text plus the user's auth token to the backend.
3. **AI Check:** Backend computes the text embedding and queries the vector DB. Suppose it retrieves 5 similar posts. The service computes cosine/text similarity with each. All scores are below threshold (e.g. max 65%).

4. **Block Registration:** Backend hashes the content (h = SHA256(text)) and calls the blockchain contract function `registerContent(h, [])` (an empty array since no high-similarity found). The user signs this TX with MetaMask.
5. **On-Chain Record:** The smart contract stores (author=wallet, hash=h, time=now, similarRefs=[]). It emits `ContentRegistered` with a new content ID, say `42`.
6. **Confirmation:** The frontend shows the user: "Your post is original! Block ID: 42 (TxHash: 0xabc…)". The data is now immutable on-chain.

Security and Trust

- **Tamper-Proof Evidence:** By design, no post can be erased or altered once on-chain. Even if hackers corrupt our servers or DB, they cannot change the blockchain history. This is why data on the ledger is "ideal for judicial evidence".
- **Authentication:** MetaMask ensures only the legitimate wallet holder can register a post under their name. Each transaction is signed by the user's private key, guaranteeing authorship authenticity.
- **Privacy:** We never store plain text on the public chain. Only content hashes or IPFS CIDs appear on-chain. This protects user privacy while still proving content integrity.
- **Decentralized Governance:** No central admin is needed to vouch for a post's originality. If a post's chain record exists, any node can verify it independently. As Sarcinella *et al.* note, blockchain enables secure "transactions transcending trust" – here, "transaction" means recording the content's authenticity.
- **Resistance to Fraud:** If someone tries to post content plagiarized from a chain-registered post, our AI detection will catch the similarity and block it (or flag it). If they somehow bypass AI, the community can always check the blockchain: the earlier entry stands as proof the content already existed. This disincentivizes copying.

Advantages of This Approach

- **Immutability & Transparency:** Every post's authenticity is auditable by anyone. In contrast to opaque central databases, the blockchain record is fully transparent.
- **Decentralization:** No single point of failure or control. Even if our service is offline, the content ledger survives on the distributed network of nodes.
- **Strong Plagiarism Detection:** Using transformer embeddings and semantic search addresses "limitations of traditional tools" (like missing paraphrases).

- **Legal Proof:** In contexts like academic writing or creative works, having a decentralized timestamp can serve as strong evidence of originality. The literature confirms that immutable blockchain records support rights protection.
- **User Empowerment:** Creators on Konthokosh gain verifiable proof of their work. They can share with confidence, knowing the system automatically enforces originality.

Future Work and Enhancements

- **Scaling:** As the content corpus grows, we may shard the vector index or use hierarchical retrieval to maintain performance.
- **Smart Contract Features:** We could add on-chain reputation tokens, reward systems, or decentralized moderation via DAO.
- **Extended Content Types:** The same framework could be adapted to images or code by using suitable embeddings and hashing techniques.
- **Legal Integration:** We might provide an official certificate (via smart contract) that copyright holders can use in legal processes, similar to “copyright certificate” downloads in related systems.

Conclusion

Konthokosh demonstrates how **AI and blockchain together** can create a secure, decentralized platform for sharing original content. By combining semantic similarity search with on-chain notarization, we ensure that user-generated stories and poems can be published “without fear of plagiarism.” No existing tool today offers this exact combination, making our solution novel. Citations show that blockchain-based notarization and anti-plagiarism is a promising frontier. Konthokosh’s architecture ensures content integrity: documents are semantically checked for duplicates, and originals are immutably stamped on a public ledger. This creates a trustless ecosystem where creative authorship is transparently verified and protected.

Sources: This design synthesizes recent research on blockchain notarization and content integrity, transformation-based plagiarism detection, and decentralized storage architectures. The Merkle-of-ideas is supported by prior systems for digital copyright, adapted here for a social platform.