Blockchain-Based Framework for Secure Electronic Health Records with Machine Learning for Disease Risk Prediction

Rajan R[1], Lin Eby Chandra J[2]
[1,2]*Department of Computer Science and Engineering Jaya Engineering College, Anna University, Chennai, India*

***Abstract*—Medical records in most healthcare settings are scattered across the many hospitals, laboratories, and clinics a person visits over a lifetime. The practical results are duplicated tests, missing context at the point of care, and patients who cannot easily assemble their own history when they need it most. This paper presents MediChain, a medical history management platform that returns ownership of records to the patient while keeping the underlying documents both confidential and independently verifiable. Each record file is encrypted on the server before it leaves the backend and is stored on the InterPlanetary File System (IPFS); only a compact reference, namely a content identifier together with record metadata, is anchored on an Ethereum-compatible smart contract. The contract governs who may write and read a given patient's records: a doctor must first be verified by an administrator and then explicitly granted access by the patient before any record can be added, and the patient can revoke that access at any time. Alongside this integrity layer, a deliberately decoupled machine-learning service estimates disease-risk probability for diabetes, heart disease, chronic kidney disease, and liver disease from structured laboratory values, returning a calibrated risk band rather than a clinical diagnosis. We evaluate the smart contract with a twenty-case unit-test suite that exercises every authorised and unauthorised access path, we report held-out performance for the four risk models (receiver-operating-characteristic area under the curve from 0.76 to 0.95), and we validate the complete record workflow including server-side encryption and cryptographic integrity checking. The architecture is phased so that the core clinical application, the risk-scoring service, and the blockchain layer can each be switched on independently, which keeps a working system demonstrable at every stage of the build.**

***Index Terms*—Blockchain, Electronic Health Records, InterPlanetary File System (IPFS), Smart Contracts, Role-Based Access Control, Machine Learning, Disease-Risk Prediction, Data Integrity, FastAPI, Solidity.**

I. INTRODUCTION

A patient's medical history is rarely held in one place. Prescriptions are written at a clinic, blood work is run at a separate laboratory, an imaging centre keeps the scans, and a hospital admission generates yet another set of notes. Each of these custodians stores its slice of the record in its own system, under its own format and access rules. When the same patient later presents elsewhere, the treating clinician usually starts with an incomplete picture, which leads to repeated tests, slower diagnosis, and avoidable risk. Recent surveys of healthcare data interoperability describe exactly this pattern of isolated silos and document how the resulting blind spots delay care and inflate cost [1], [2]. Moving the centre of gravity from the institution to the patient, so that the individual holds a verifiable and portable copy of their own history, is therefore a worthwhile design goal rather than a purely technical one.

Three structural problems recur when one tries to build such a system. First, ownership is institutional rather than personal: the data lives inside provider databases that the patient can neither carry with them nor authorise others to read on their own terms. Second, simply pooling records in a shared database does not by itself make them trustworthy, because whoever controls that database can in principle alter or remove an entry, and a single central store is an obvious target. Third, the clinical value locked inside accumulated

laboratory results is seldom surfaced; the numbers sit in files and are read once, even though they often carry early signals of rising disease risk. A credible platform has to address all three at once, which means combining patient-controlled access, tamper-evident storage, and analytics in a single coherent design.

This paper introduces MediChain, a medical history management system built to meet those three requirements together. Record files are encrypted before storage and pinned to IPFS, while a Solidity smart contract on an Ethereum-compatible network holds only the content reference and the access rules. The contract treats the patient's wallet as the authority over their own records: it records which doctors are verified, which doctors a patient has authorised, and a reference for every record that an authorised doctor adds. A separate machine-learning service reads the structured laboratory values attached to test reports and reports a disease-risk probability for four common conditions. The clinical application, the risk service, and the on-chain layer are kept loosely coupled and are individually toggled by configuration, so the system degrades gracefully and presents as a complete application even when the optional layers are switched off.

The contributions of this work are fourfold:

- We present a patient-centric architecture in which the smart contract is the source of truth for record references and access permissions, while the application database is treated only as a fast-query mirror, so record integrity never depends on trusting the backend.
- We design and implement a Solidity access-control contract whose write and read paths are gated by a verified-doctor check combined with patient-granted consent, and we bind each application user to a deterministic wallet so that the on-chain transaction sender is the actual person performing the action rather than a single shared relay key.
- We integrate a decoupled disease-risk service for diabetes, heart, kidney, and liver conditions that returns a probability and a risk band rather than a diagnosis, a framing chosen for both ethical and regulatory defensibility.
- We validate the system across three dimensions, namely a complete unit-test suite over the contract's access logic, held-out evaluation of the four risk models, and an end-to-end test of the encrypt, store, anchor, and verify workflow, and we report the outcomes honestly together with the limitations that remain.

The remainder of the paper is organised as follows. Section 2 reviews related work on blockchain based health-record management, smart-contract access control, and machine learning for disease-risk prediction. Section 3 describes the methods, covering the layered architecture, the decomposition into subsystems, the component design, and the implementation. Section 4 reports the unit, model, and integration results. Section 5 discusses what the results mean and states the limitations plainly. Section 6 concludes and outlines future work.

II. RELATED WORK

2.1 Blockchain and Decentralised Storage for Health Records

Using a blockchain as the trust anchor for electronic health records has become an active research line, and a consistent design pattern has emerged across recent work: keep the bulky, sensitive document off the chain and place only a hash or content reference on it. Frameworks that pair a blockchain ledger with IPFS report that this split preserves immutability and auditability while keeping cost and latency manageable, since the chain never carries the file itself [3], [4]. Patient-centred variants extend the idea by letting the individual grant or withdraw a provider's access, which shifts control away from the institution and toward the data subject [5], [6]. Measurements of on-chain versus off-chain storage support the same conclusion quantitatively: moving file data to IPFS and retaining only a reference on the ledger cuts on-chain storage consumption dramatically while still allowing any later tampering to be detected through a hash mismatch [10], [11]. MediChain follows this established split but is explicit about an often glossed-over detail, namely what the on-chain reference should be when a deployment chooses not to run IPFS; in that mode the system anchors the file's own content hash, so the integrity guarantee holds either way.

2.2 Smart-Contract Access Control

A second body of work focuses on encoding the access policy itself as a smart contract. Patient consent expressed on Ethereum yields authorisations that are tamper-resistant and revocable, and role-based and

attribute-based models have been layered on top to satisfy regulatory expectations around consent and audit [7], [9]. More recent frameworks combine several contract types, for example separate user-management, policy, and access-control contracts, and report improved throughput and lower deployment cost on private Ethereum networks [8]. The common thread is that the contract, not an application server, decides who may act. MediChain adopts the same principle in a deliberately compact form: a single contract holds the verified-doctor set, the per-patient consent mapping, and the record references, and it refuses any write from a doctor who is not both verified and currently authorised by the patient. Keeping the policy small makes it straightforward to test exhaustively, which we exploit in Section 4.

2.3 Machine Learning for Disease-Risk Prediction

Independent of the storage question, supervised models trained on routine clinical and laboratory variables have repeatedly been shown to flag elevated disease risk. Gradient-boosted trees, and XGBoost in particular, tend to lead recent comparative studies for cardiovascular and metabolic risk, with regularised logistic regression remaining a strong and interpretable baseline [14], [15], [16], [17]. Public benchmark datasets such as the Pima Indians Diabetes database and the UCI Cleveland heart, chronic kidney disease, and Indian liver patient collections continue to serve as the standard proving ground for this class of model [13], [19]. MediChain does not attempt to advance the modelling state of the art; it instead treats these well-understood models as a reusable service that consumes the same laboratory values already captured with each test report, and it presents their output as a graded risk indication that is kept entirely separate from the record-integrity layer.

III. METHODS

This section sets out the design of MediChain. It begins with the layered architecture, then decomposes the application into its functional subsystems, details the component design of the parts that carry the security guarantees, and closes with the implementation and deployment choices that realise the design.

3.1 System Architecture

MediChain is organised as a five-part architecture that separates presentation, application logic, the analytics service, the decentralised integrity layer, and persistence. Each part communicates with its neighbours through narrow, well-defined interfaces so that any one of them can be replaced or disabled without disturbing the rest.

The presentation layer is a single-page React application that renders three role-specific dashboards, one each for the administrator, the doctor, and the patient. All views share a common navigation shell and talk to the backend exclusively over a REST interface protected by JSON Web Token authentication, so the browser never holds a private key or contacts the chain directly.

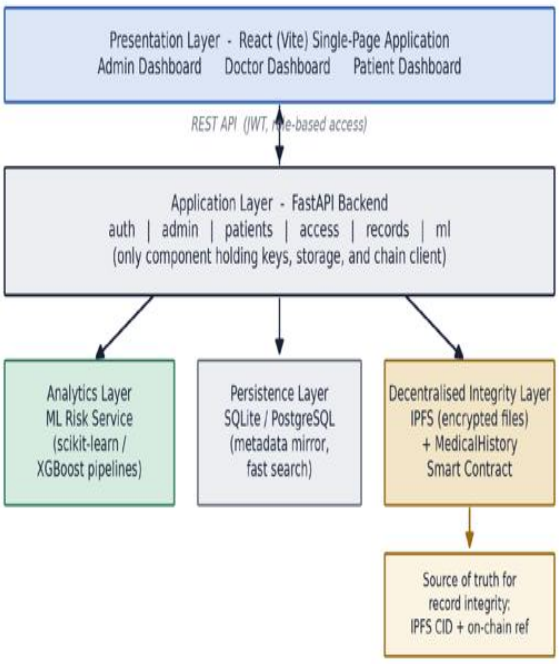
The application layer is a FastAPI service that owns routing, request validation, role enforcement, and the orchestration of every action. It exposes grouped endpoints for authentication, administration, patient management, access control, records, and risk scoring. This layer is the only component that touches encryption keys, the storage backend, and the chain client, which keeps the trust boundary in one place.

The analytics layer is the machine-learning service. It is decoupled from the integrity layer entirely and operates only on structured laboratory values, returning a probability and a risk band for a requested condition. Because it shares nothing with the storage and chain code, it can be enabled, disabled, or retrained on its own.

The decentralised integrity layer comprises encrypted file storage on IPFS through a pinning service and the MedicalHistory smart contract on an Ethereum-compatible network. Together they form the tamper-evident record of what was stored, by whom, for which patient, and when. This layer is the source of truth for record integrity and access rights.

The persistence layer is a relational database accessed through an object-relational mapper. It mirrors user accounts, record metadata, current access state, and saved risk assessments so that the application can search and list quickly. It is explicitly not the source of truth for record integrity; that role belongs to the IPFS content identifier and the on-chain reference.

Figure 1. MediChain five-layer system architecture.



3.2 Decomposition into Subsystems

The application layer is decomposed into four subsystems, each carrying one functional concern.

3.2.1 Identity and Authentication Subsystem

This subsystem registers users, authenticates them, and assigns each a role of administrator, doctor, or patient. Passwords are stored only as bcrypt hashes, and a successful login returns a signed token that carries the user identifier and role and expires after a configured interval. Every protected endpoint resolves the caller from that token and checks the role before doing any work. Administrators onboard hospitals and create doctor accounts, and a doctor remains unverified, and therefore unable to write records, until an administrator verifies them.

3.2.2 Records and Storage Subsystem

This subsystem handles the upload, retrieval, decryption, and integrity verification of record files. A record carries its type (prescription, test report, or scan report), descriptive metadata, the storage location, the plaintext content hash, and, for test reports, the structured laboratory values that the analytics layer later consumes. Files are encrypted before storage and decrypted only when an authorised viewer downloads them.

3.2.3 Access-Control and Chain Subsystem

This subsystem expresses patient consent. A patient grants or revokes a named doctor, and the change is written both to the application database and, when the chain layer is active, to the smart contract under the patient's own wallet. The same subsystem mirrors administrator actions, namely doctor verification and patient registration, onto the contract. Application-level checks shadow the on-chain rules so that the user interface can enforce them immediately without a round-trip to the chain.

3.2.4 Disease-Risk Subsystem

This subsystem loads the trained pipelines, validates an incoming feature set against the model's declared schema, and returns a probability and a risk band. It can optionally persist the result as a risk assessment linked to the patient and, where relevant, to the originating record, which produces a longitudinal trail of risk over time.

3.3 Component Design

This subsection describes the internal structure of the two components that carry the security and analytics guarantees: the smart contract and the record-ingestion pipeline. The risk service is summarised where its interface meets the rest of the system.

3.3.1 The MedicalHistory Smart Contract

The contract maintains four pieces of state. A mapping from patient identifier to a list of record references records what has been stored. A nested mapping from patient identifier to doctor address to a boolean holds the consent matrix. A mapping from doctor address to a boolean holds the set of verified doctors. A mapping from patient identifier to wallet address binds each patient to the wallet that speaks for them. The patient identifier itself is a thirty-two-byte value derived by hashing a stable per-patient string, so the on-chain identifier never reveals a real-world identity.

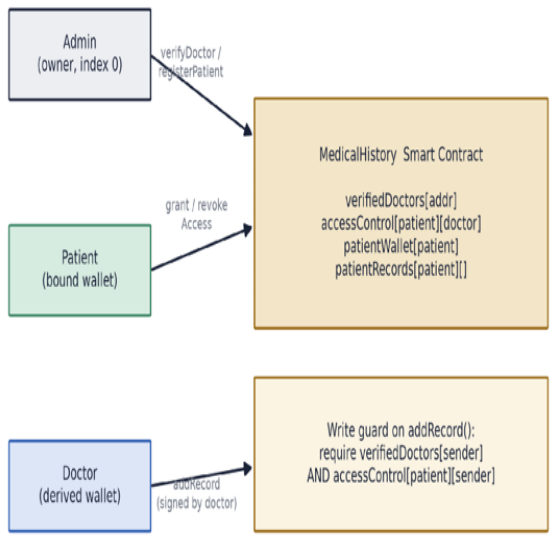
The contract exposes a small set of functions. The contract owner, who is the administrator, calls verifyDoctor to admit a doctor and registerPatient to bind a patient to a wallet. A patient calls grantAccess and revokeAccess to manage consent, and the contract checks that the caller is that patient's bound wallet (or the owner) before applying the change. A verified and authorised doctor calls addRecord to append a reference, and read access through getPatientRecords and getRecordCount is restricted to the owner, the patient, or a doctor the patient currently authorises.

Every state-changing call emits an event, which yields an immutable, independently auditable trail of who did what and when. The central guard is the write path, where two conditions must both hold:

```
function addRecord(bytes32 patientId, string calldata cid,
                   string calldata recordType, uint256 timestamp) external {
  require(verifiedDoctors[msg.sender], "Doctor not verified");
  require(accessControl[patientId][msg.sender], "Access not granted");
  patientRecords[patientId].push(
    RecordReference(msg.sender, cid, recordType, timestamp));
  emit RecordAdded(patientId, msg.sender, cid, recordType, timestamp);
}
```

Because the check reads msg.sender, the doctor's own wallet has to sign the transaction; a backend cannot impersonate a doctor without that doctor's key. To make this property hold in practice without asking every clinic to manage a wallet by hand, the backend derives a deterministic wallet for each user from a hierarchical-deterministic seed at a fixed derivation path, reserving index zero for the administrator-owner. The address a user is given on the chain therefore lines up with the identity the application already knows, and the on-chain access checks are genuinely enforced rather than simulated by a single shared key.

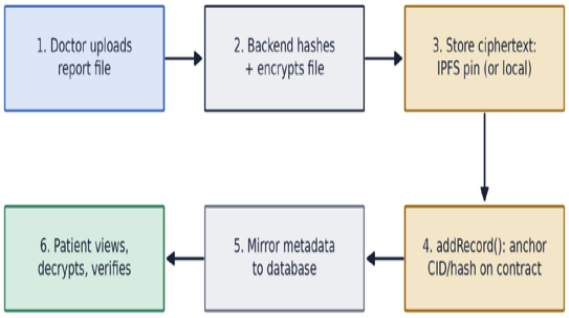
Figure 3. Smart-contract state and access-control interactions.



3.3.2 The Record-Ingestion Pipeline

When a doctor uploads a file, the records endpoint runs a fixed sequence. It first confirms that the target patient exists and that the doctor currently holds access, refusing the upload otherwise. It reads the file, computes the SHA-256 hash of the plaintext for later integrity checks, and encrypts the bytes with a symmetric key before anything is written to disk or the network. It then stores the ciphertext: if IPFS is enabled it pins the encrypted bytes and keeps the returned content identifier, and if IPFS is unavailable or disabled it falls back to encrypted local storage so the upload never fails on a transient network problem. Next, when the chain layer is active and the doctor is verified, it submits the on-chain reference through addRecord; that reference is the IPFS content identifier when IPFS is on and the file's content hash otherwise, so a verifiable anchor exists in both modes. Finally it writes the record metadata to the database for fast retrieval. On viewing, the reverse path fetches the ciphertext, decrypts it for an authorised viewer, and a dedicated verification endpoint recomputes the hash and reports whether the stored file still matches its anchored reference.

Figure 2. Data flow for adding and verifying a record.



Verification recomputes the plaintext hash from stored ciphertext and compares it with the anchored reference (CID or sha256:hash).

3.3.3 The Risk-Scoring Component

Each disease model is a single serialised pipeline that chains median imputation, standardisation, and a classifier, accompanied by a small metadata file that records the ordered feature schema, the model type, the evaluation metrics, and the sample count. At request time the service builds a single-row frame in the model's declared feature order, tolerating missing

or aliased keys by letting the imputer fill gaps, and converts the positive-class probability into a low, moderate, or high band using fixed thresholds. The output is deliberately a graded probability and never a binary diagnosis.

3.4 Implementation

3.4.1 Technology Stack

The backend is a FastAPI application written in Python, using SQLAlchemy as the object-relational mapper over SQLite by default, with a one-line switch to PostgreSQL for a production deployment. Authentication uses JSON Web Tokens with bcrypt password hashing. File confidentiality uses symmetric authenticated encryption (Fernet, which combines AES in CBC mode with an HMAC), applied before any file reaches disk or IPFS. The frontend is a React single-page application built with Vite and React Router, communicating over REST. The machine-learning models are scikit-learn and XGBoost pipelines served behind dedicated endpoints. The smart contract is written in Solidity and compiled, tested, and deployed with the Hardhat toolchain; the backend talks to it through the Web3 client library.

3.4.2 Module Implementation

The records module is implemented as a FastAPI endpoint that accepts multipart form data and executes the ingestion pipeline of Section 3.3.2. The access module exposes grant and revoke endpoints that update the database and, when enabled, mirror the change on-chain under the patient's wallet, registering the patient on first use. The administration module creates and verifies doctors and assigns each new user the next free wallet index. The risk module validates the requested condition against the available models and returns the probability and band, optionally saving the assessment. The integrity check at the heart of the verification endpoint is compact: it recomputes the plaintext hash from the stored ciphertext and compares it with the value recorded at upload time.

```
def verify(record):
    plaintext = decrypt(read_ciphertext(record))
    recomputed = sha256_hex(plaintext)
    intact = recomputed == record.file_hash
    # the anchored reference is the IPFS CID, or 'sha256:<hash>' when IPFS is off
    return intact
```

3.4.3 Deployment and Build Phasing

The system is containerised, with a Compose definition provided for the database tier, and it is designed to be built in three phases that each leave a demonstrable system behind. Phase one is the core clinical application with authentication, roles, and encrypted upload and view, which works on its own with no chain or IPFS. Phase two adds the four risk models behind their endpoints. Phase three switches storage to IPFS, deploys the contract, and wires the on-chain access and record calls. Each later phase is gated by a configuration flag, so the platform always presents as a complete application even with the optional layers off.

One deployment note is worth stating plainly because it shaped the build. The original architecture targeted the Polygon Mumbai test network, which was retired in April 2024. The identical contract now runs on a local Hardhat node by default, which is free, instant, and offline-friendly for development and evaluation, and the project ships a ready configuration for the Polygon Amoy test network so the same bytecode can be promoted to a public testnet without code changes.

IV. RESULTS

We evaluate MediChain along three axes that map to its three guarantees: the correctness of the contract's access logic, the predictive quality of the risk models, and the end-to-end behaviour of the record workflow including confidentiality and integrity. Test cases follow directly from the design in Section 3 and cover both authorised and unauthorised paths.

4.1 Smart-Contract Unit Testing

The contract is exercised by a unit-test suite written with the Hardhat, Chai, and ethers tooling, comprising twenty cases grouped by function. The suite checks not only that authorised actions succeed and emit the expected events but, just as importantly, that unauthorised actions revert: a non-owner cannot verify a doctor or register a patient, an unrelated account cannot grant access on a patient's behalf, an unverified doctor cannot write, a verified doctor without consent cannot write, and a revoked doctor immediately loses read access. Table 1 summarises the groups and outcomes. All twenty cases pass.

Function group	Cases	Property verified	Result

Deployment	1	Deployer is recorded as the contract owner	Pass
verifyDoctor	2	Only the owner verifies a doctor; others revert	Pass
registerPatient	3	Owner-only registration; duplicate registration reverts	Pass
grantAccess / revokeAccess	5	Only patient wallet or owner changes consent	Pass
addRecord	3	Write requires verified doctor and active consent	Pass
Read gating	6	Read limited to owner, patient, or authorised doctor	Pass
Total	20	All authorised and unauthorised paths	20 / 20 pass

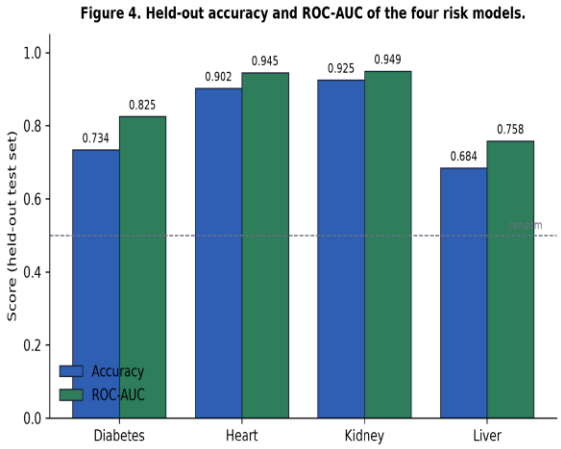
Table 1: Smart-contract unit-test summary.

4.2 Disease-Risk Model Evaluation

Each model was trained on an eighty-twenty stratified split with a fixed random seed, using median imputation and standardisation ahead of the classifier, and was evaluated on the held-out portion. Diabetes and chronic kidney disease use regularised logistic regression with balanced class weighting; heart and liver disease use gradient-boosted trees. Table 2 reports accuracy and the area under the receiver-operating-characteristic curve along with the sample size of each public dataset. The kidney and heart models are the strongest, with areas under the curve of 0.95 and 0.94, the diabetes model is solid at 0.82, and the liver model is the weakest at 0.76, which is consistent with the known difficulty of the Indian liver patient dataset and which we treat openly as a limitation rather than smoothing over.

Condition	Model	Samples	Accuracy	ROC-AUC
Diabetes	Logistic Regression	768	0.734	0.825
Heart disease	Gradient-Boosted Trees	303	0.902	0.945
Kidney disease	Logistic Regression	400	0.925	0.949
Liver disease	Gradient-Boosted Trees	583	0.684	0.758

Table 2: Held-out performance of the four disease-risk models.



4.3 Integration Testing

Integration testing exercised the full clinical loop end to end. An administrator created a hospital, created a doctor, and verified that doctor; a patient account was created and granted the doctor access; the doctor uploaded a test report carrying laboratory values; the patient listed and viewed the record; a risk score was computed from the attached values and saved against the patient; and finally the patient revoked the doctor's access and confirmed that subsequent reads and writes were refused. The tests confirmed that state moves correctly between subsystems, that access decisions taken in one place are honoured everywhere, and that the database mirror stays consistent with the actions taken.

4.4 Confidentiality and Integrity Validation

Two properties specific to this platform were checked directly. For confidentiality, the stored artefact for every uploaded file was inspected and confirmed to be ciphertext, so a leaked content identifier or a stolen file on its own exposes nothing without the key. For integrity, the verification endpoint was run against intact files, which it reported as matching, and against a deliberately altered stored file, which it correctly flagged as a hash mismatch. This demonstrates that the anchored reference, whether an IPFS content identifier or a content hash, detects tampering after the fact as intended.

4.5 Summary of Outcomes

Across the three axes the system behaved as designed. The twenty contract tests passed, including every negative case; the four risk models produced areas under the curve from 0.76 to 0.95 on held-out data; and the integration and integrity checks confirmed that encryption, anchoring, access enforcement, and tamper detection work together over the complete workflow. The one result we flag rather than celebrate is the liver model, whose moderate score reflects the dataset and motivates the future work in Section 6.

V. DISCUSSION

The results support the central design claim, namely that record integrity and access control can be made to rest on a small, exhaustively tested contract rather than on trust in the application server. Three points are worth drawing out against the landscape surveyed in Section 2.

First, treating the contract as the source of truth and the database only as a mirror is what makes the integrity guarantee meaningful. Because the verification endpoint recomputes the hash from the stored ciphertext and compares it with the anchored reference, a silent change to the database or the file is detectable, which is precisely the property that a shared central database cannot offer on its own. This mirrors the on-chain reference plus off-chain storage pattern reported in recent frameworks [3], [4], [10], and it adds the explicit fallback of anchoring a content hash when IPFS is not in use, so the guarantee does not silently disappear in a lighter deployment.

Second, binding each user to a deterministic wallet means the contract's checks are real. Many prototypes route every transaction through one backend key, which collapses the on-chain identity of doctor, patient, and administrator into a single signer and quietly defeats the access model they describe. By signing each action with the acting user's own derived key, MediChain keeps msg.sender meaningful, so the verified-doctor and patient-consent requirements are enforced by the chain and not merely reflected by the user interface.

Third, keeping the risk service decoupled is a deliberate safety choice as much as an architectural one. The models never touch the integrity layer and never return a diagnosis; they return a graded probability that a clinician interprets. This keeps the platform on the defensible side of the line between a decision-support indicator and a regulated diagnostic device, which matters for both ethics and eventual deployment.

Several limitations remain, and we state them plainly rather than leaving them implicit. The contract currently runs on a local development network by default; although a public-testnet configuration is provided, a sustained public deployment with gas and latency measurement is still future work, a situation forced in part by the retirement of the originally targeted test network. File confidentiality presently uses a single master key rather than per-patient keys, which is adequate for the foundation build but should become per-patient key management before any real use. IPFS is optional, and the default local-storage fallback, while convenient, is not decentralised, so the full integrity story holds only when IPFS is enabled. The default database is SQLite, which is fine for evaluation but would move to PostgreSQL in production. The risk models are trained on modest public datasets and are not clinically validated; the liver model in particular is only moderate. Finally, the application database mirror can in principle drift from the chain, which we mitigate with best-effort synchronisation and the independent verification endpoint but do not yet reconcile automatically.

VI. CONCLUSION AND FUTURE WORK

6.1 Conclusion

This paper presented MediChain, a medical history management platform that combines patient-owned access control, encrypted decentralised storage, and machine-learning disease-risk scoring in one coherent

system. The smart contract holds the verified-doctor set, the per-patient consent matrix, and a reference for every record, and it refuses any write that is not from a verified doctor the patient currently authorises. Record files are encrypted before storage and anchored either by an IPFS content identifier or by a content hash, so tampering is detectable in either deployment mode. A decoupled service scores risk for four common conditions as a graded probability rather than a diagnosis. The evaluation showed a fully passing twenty-case contract test suite, held-out model areas under the curve from 0.76 to 0.95, and a verified end-to-end workflow in which encryption, anchoring, access enforcement, and tamper detection operate together. Just as important for a buildable system, the phased and flag-gated design keeps a working application demonstrable at every stage.

6.2 Future Work

Five directions follow naturally from the limitations above. First, promote the contract to a public test network such as Polygon Amoy and report real gas, throughput, and latency figures under load. Second, replace the single master key with per-patient key management so that confidentiality survives a broader threat model. Third, add an emergency-access mechanism with explicit patient rules and a fully audited break-glass path, so urgent care is possible without weakening day-to-day consent. Fourth, strengthen the risk service with larger and more representative datasets, probability calibration, and trend detection across a patient's successive laboratory results, with particular attention to the weaker liver model. Fifth, add automatic reconciliation between the on-chain state and the database mirror, together with optical character recognition to digitise paper prescriptions and wider interoperability with external health systems.

Declarations

Funding
No funding was received for conducting this study.

Conflicts of Interest

The authors declare no conflict of interest.

Data Availability

The disease-risk models were trained on publicly available benchmark datasets (the Pima Indians Diabetes database and the UCI Cleveland heart, chronic kidney disease, and Indian liver patient datasets). The MediChain source code, smart contract, tests, and deployment configuration are available from the authors on reasonable request.

Author Contributions

Rajan R: Conceptualisation, system design, implementation, deployment, writing. J. Lin Eby Chandra: Supervision, methodology validation, manuscript review and editing.

Use of Generative AI

The authors used a generative AI assistant to support language refinement, structural organisation, and proofreading of the manuscript. All technical content, system design, implementation, testing, and conclusions are the original work of the authors. AI-generated suggestions were reviewed and verified by the authors, who retain full intellectual responsibility for the manuscript.

ACKNOWLEDGEMENTS

The authors thank the Department of Computer Science and Engineering, Jaya Engineering College, for the support extended throughout this project.

REFERENCES

[1] From Data Silos to Health Records Without Borders: A Systematic Survey on Patient-Centered Data Interoperability, Information, vol. 16, no. 2, art. 106, 2025.
[2] Bridging Healthcare Data Silos for Better Patient Outcomes: an analysis of fragmentation and integration barriers in modern health systems, industry technical report, 2024.
[3] A Blockchain-IPFS Framework for Secure, Scalable, and Interoperable Healthcare Data Management, SN Computer Science, Springer, 2025.
[4] Toward Blockchain-Based Electronic Health Record Management with Fine-Grained Attribute-Based Encryption and Decentralized Storage Mechanisms, Scientific Reports, vol. 15, 2025.

[5] A Blockchain-Based Smart Healthcare System for Data Protection, peer-reviewed open-access article, 2025.
[6] Ethereum Blockchain for Electronic Health Records: Securing and Streamlining Patient Management, peer-reviewed open-access article, 2024.
[7] S. Kalita et al., Designing Efficient Patient-Centric Smart Contracts for Healthcare Ecosystems with Access Control Capabilities, Security and Privacy, vol. 7, Wiley, 2024.
[8] MedAccessX: A Blockchain-Enabled Dynamic Access Control Framework for IoMT Networks, peer-reviewed open-access article, 2024.
[9] A Medical Big Data Access Control Model Based on Smart Contracts and Risk in the Blockchain Environment, Frontiers in Public Health, vol. 12, art. 1358184, 2024.
[10] Research on an On-Chain and Off-Chain Collaborative Storage Method Based on Blockchain and IPFS, Future Internet, MDPI, 2025.
[11] J. Benet, IPFS: Content Addressed, Versioned, Peer-to-Peer File System, arXiv:1407.3561, 2014.
[12] G. Wood, Ethereum: A Secure Decentralized Generalized Transaction Ledger, Ethereum Project Yellow Paper, 2014 and subsequent revisions.
[13] Pima Indians Diabetes Mellitus Classification Based on Machine Learning Algorithms, peer-reviewed open-access article, 2022.
[14] A Systematic Review of Machine Learning in Heart Disease Prediction, peer-reviewed open-access article, 2025.
[15] Cardiovascular Disease Prediction Using Machine Learning: A Comparative Analysis, arXiv:2507.21898, 2025.
[16] Development and Validation of a Machine Learning Model for Cardiovascular Disease Risk Prediction in Type 2 Diabetes Patients, Scientific Reports, 2025.
[17] T. Chen and C. Guestrin, XGBoost: A Scalable Tree Boosting System, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.
[18] F. Pedregosa et al., Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, vol. 12, pp. 2825 to 2830, 2011.
[19] D. Dua and C. Graff, UCI Machine Learning Repository (Cleveland Heart Disease, Chronic Kidney Disease, and Indian Liver Patient datasets), University of California, Irvine, School of Information and Computer Sciences.
[20] M. Jones, J. Bradley, and N. Sakimura, JSON Web Token (JWT), RFC 7519, Internet Engineering Task Force, 2015.