

A Two-Tier Blockchain Architecture for Verifiable Explainable-AI Audit Trails: Individual and Merkle-Batched Commitment of SHAP-Based Fraud-Detection Decisions

Kavindu Karunarathne

Lecture, Department of Computer and Data Science, York St John University, London, United Kingdom.

E-mail: k.karunarathne@yorks.ac.uk

Abstract

Machine learning systems that make automated decisions, including fraud flags, loan denials, and safety interlocks, typically generate an explanation (for example via SHAP) in the same trust domain as the model itself, leaving no independent proof that the explanation shown to an auditor is the one the model actually produced at decision time. This research presents and empirically evaluates a blockchain-anchored explainable-AI (XAI) audit-trail architecture that closes this gap by committing a canonical SHA-256 hash of each decision-and-explanation record to an Ethereum-compatible ledger, while keeping the full record, including the SHAP explanation, off-chain. The architecture is implemented end to end on a real fraud-detection pipeline consisting of a LightGBM classifier trained on the IEEE-CIS Fraud Detection dataset (590,540 transactions), a SHAP TreeExplainer layer producing per-transaction feature attributions, a canonical JSON serialization and hashing procedure, and two complementary Solidity smart contracts: one committing each audit record individually, and the other committing batches of records via a Merkle root. Using 500 audit records, the schemes are benchmarked on an Ethereum-compatible network across batch sizes of 10 to 500 records and extended with synthetic hashes to 1,000, 2,500, and 5,000 records. Individual commitment registers a record in 6.3 to 17.4 ms at 139,300 gas per record. Merkle-root batching reduces

on-chain gas consumption by 91.6% at 10 records to 99.98% at 5,000 records, with reduction factors rising from roughly 12-fold to nearly 5,982-fold. Tamper-detection trials correctly flagged altered records in 100% of cases. The work contributes a reproducible, empirically costed reference architecture for verifiable AI explanation integrity, and a direct, quantified comparison of two on-chain commitment strategies across three orders of magnitude of batch size that prior blockchain-XAI proposals have not benchmarked head to head.

Keywords: Explainable AI, SHAP, Smart Contracts, Merkle Tree, Fraud Detection, Gas Optimization, Non-Repudiation, Ethereum.

1. Introduction

Automated decision systems increasingly determine outcomes that matter: whether a transaction is blocked as fraudulent, whether a loan application is declined, whether a manufacturing robot's action is permitted to proceed. Explainable AI (XAI) techniques such as SHAP (SHapley Additive exPlanations) [1] are now routinely used to accompany such decisions with a feature-level justification, and regulatory instruments such as the European Union's General Data Protection Regulation have been read as creating an effective right to explanation for individuals affected by automated decisions [2]. In almost every deployed system, however, the explanation is generated, stored, and later retrieved from within the same trust domain as the model that produced the decision, typically the deploying organisation's own database. If a decision is later disputed, for example a customer challenges a declined transaction, a regulator audits a lending model, or an insurer investigates a blocked payment, there is no independent mechanism to prove that the explanation now being shown was the one actually computed at decision time, rather than one regenerated, adjusted, or selectively retained after the fact.

This study addresses this gap by developing and empirically evaluating a blockchain-anchored commitment architecture for AI decision explanations. The study investigates a specific, empirically tractable question: whether a compact cryptographic commitment to an AI decision-and-explanation record, anchored on a public-style, tamper-evident ledger [3] at the moment of decision, can provide a practical, independently verifiable audit trail, and at what on-chain cost. This question is addressed empirically rather than only at the design level:

the architecture is implemented in full, and its cost is measured directly across a wide range of operating scales.

The system implemented and evaluated in this study operates on a real, non-trivial machine learning task: fraud detection on the IEEE-CIS Fraud Detection dataset [4], a 590,540-transaction benchmark released by Vesta Corporation and the IEEE Computational Intelligence Society. A LightGBM gradient-boosted classifier is trained on a chronological split of the data, its decision threshold is tuned by validation-set F1 search, and each flagged transaction is explained using SHAP's exact TreeExplainer. Every decision-and-explanation pair is serialized into a canonical, deterministically ordered JSON record and hashed with SHA-256. Two independent Solidity smart contracts then commit these hashes on an Ethereum-compatible chain: one registers each record's hash individually, and the other registers a single Merkle root representing an entire batch of records, with per-record inclusion proved via a Merkle proof rather than a per-record transaction.

Both schemes are benchmarked on 500 real audit records, genuine SHAP explanations of genuine held-out fraud predictions rather than synthetic placeholders, across five batch sizes (10, 50, 100, 250, 500), each repeated five times for statistical stability, and subsequently on synthetic but structurally identical commitment hashes extended to batch sizes of 1,000, 2,500, and 5,000 to establish whether the observed cost trends persist at higher decision volumes. On-chain registration latency, verification latency, gas consumption, throughput, and tamper-detection accuracy are measured for both schemes, yielding an explicit trade-off between the two: the individual scheme offers a single stored hash and instant equality-check verification at a flat, non-trivial per-record gas cost, while the Merkle scheme offers a near-flat, vanishingly small per-record cost at the expense of a longer, logarithmically growing proof and a delay until a batch is closed.

The contributions of this research study are fourfold. First, a complete, working reference architecture for blockchain-committed XAI audit trails, from model training through SHAP explanation, canonical hashing, and on-chain commitment, implemented rather than only specified, and presented with an explicit architecture diagram and a formal algorithmic specification of its tamper-detection procedure. Second, an empirical, reproducible cost model, given in closed mathematical form, comparing individual and Merkle-batched on-chain

commitment of the same real audit records, reporting gas, latency, and throughput as a function of batch size with repeated-trial statistics across three orders of magnitude, from 10 to 5,000 records. Third, a demonstrated tamper-detection capability, in which single-field alterations to a stored decision record are shown to break the SHA-256 commitment with certainty at every tested scale, confirming the core non-repudiation property the architecture is designed to provide. Fourth, an explicit identification of the specific insufficiencies in prior blockchain-XAI audit proposals that this architecture and evaluation address.

2. Related Works

2.1 Blockchain-Anchored AI Audit Trails

The idea of anchoring AI decision provenance to a blockchain ledger has been proposed in several recent works, generally at the level of architecture and simulation rather than benchmarked implementation. Malhotra et al. [5] proposed storing XAI decision trails using a combination of IPFS for off-chain content and Ethereum for on-chain commitment, establishing the basic pattern of keeping bulky explanation content off-chain and anchoring only a reference on-chain, the same separation of concerns adopted in this study. Kulothungan [6] proposed a permissioned-ledger architecture for logging AI inference provenance in Internet of Things (IoT) deployments, including inputs, model parameters, and outputs, to support regulatory traceability. Sachan and Liu [7] combined XAI-derived reasoning with blockchain auditing and generative-AI-assisted correspondence for pre-litigation legal decisions. Akther et al. [8] provide a survey-level treatment of blockchain as a general-purpose AI-transparency platform, arguing for the pattern without benchmarking specific commitment strategies against one another.

2.2 Gas Optimisation via Merkle Batching

Independently of the AI-auditability literature, a substantial body of Ethereum engineering research establishes storage and batching optimisation as effective means of reducing smart contract gas cost. Masla et al. [9] demonstrated that systematic gas-optimisation techniques applied to a Solidity smart contract, including batch processing of transactions, meaningfully reduce deployment and invocation cost, with further reduction achievable as the number of batched transactions increases. Mizrahi et al. [10] studied Merkle tree

organisation strategies for shortening blockchain transaction proofs, confirming that proof length, and therefore verification cost, grows logarithmically with the number of committed items when a single root is used to represent many leaves. The general pattern of storing only a single root on-chain and proving individual membership with a logarithmic-length proof is well understood to reduce gas cost substantially when many items would otherwise require independent transactions. This paper's measured reduction, 91.6% at a batch of 10 rising to 99.98% at a batch of 5,000, is consistent with, and at the upper end of, this literature, and is here measured directly on a concrete XAI audit-record workload rather than estimated from general-purpose batching benchmarks.

2.3 Research Gap and Positioning of This Work

The works reviewed are insufficient for practical deployment decisions in three specific, identifiable respects, each of which this paper addresses directly. First, the architectural proposals of Malhotra et al. [5], Kulothungan [6], and Sachan and Liu [7] are validated through illustrative examples, small-scale demonstrations, or simulation rather than a benchmarked implementation on a real, non-trivial machine learning pipeline; consequently, the practical on-chain cost of any of these designs, in gas, latency, or throughput, is not established, and a practitioner cannot determine from this literature whether the proposed architecture is affordable at production decision volumes. Second, none of this prior work compares alternative on-chain commitment strategies against one another under matched conditions on the same underlying data; a practitioner choosing between, for example, committing each decision individually or batching many decisions under a single root therefore has no empirical basis for that choice, only architectural description of one option in isolation. Third, the survey-level treatment of Akther et al. [8] argues for the general pattern of blockchain-anchored AI transparency without establishing how commitment cost scales with the volume of decisions an organisation must audit, which is precisely the parameter that determines whether such an architecture remains deployable as decision volume grows from a pilot of tens of records to a production workload of thousands.

This paper addresses all three insufficiencies. It implements the full pipeline, comprising a real, non-synthetic gradient-boosted fraud model, real SHAP explanations, canonical hashing, and two working Solidity contracts, rather than describing an architecture at the design level,

closing the first gap. It benchmarks two alternative on-chain commitment strategies against each other, with repeated trials, on the same underlying audit records, rather than evaluating a single scheme against an unspecified baseline, closing the second gap. And it reports concrete, reproducible cost figures (gas, latency, throughput, tamper-detection rate) as an explicit function of batch size across three orders of magnitude, from 10 to 5,000 records, closing the third gap and providing practitioners with a usable cost model for choosing between per-record and batched commitment at a given decision volume.

3. System Architecture

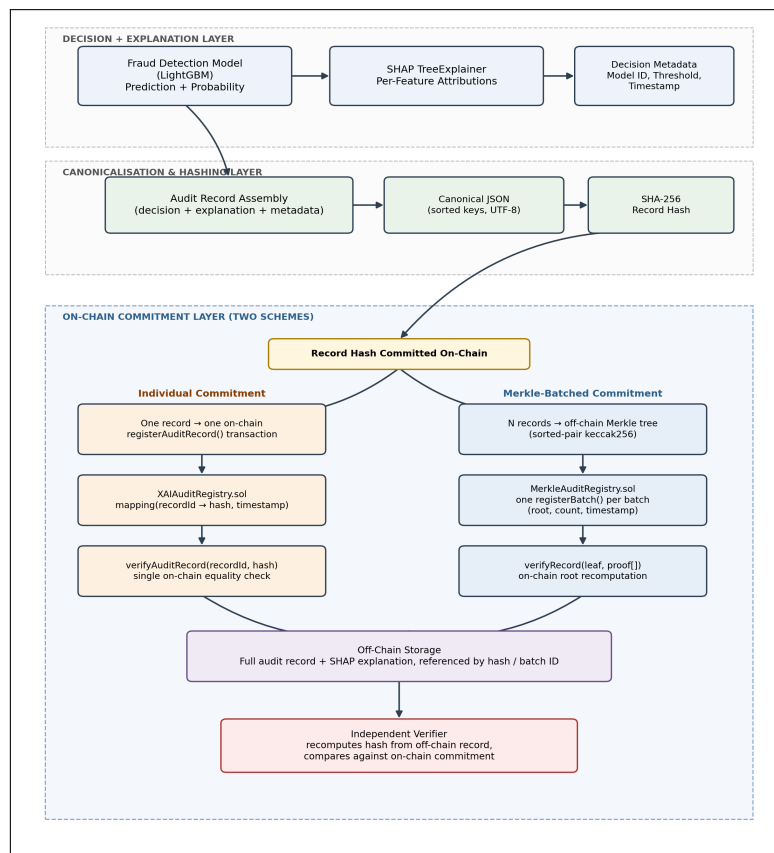


Figure 1. System architecture

The architecture separates the system into four layers, each independently replaceable: a decision layer (the fraud-detection model), an explanation layer (SHAP), a canonicalisation-and-hashing layer, and an on-chain commitment layer, the last implemented in two alternative forms. Figure 1 shows the complete data flow from decision through explanation, canonicalisation, on-chain commitment under both schemes, off-chain storage, and independent verification. This section is the single location in the paper where the mechanics of both commitment schemes are

described in full; Sections 5 and 6 refer back to the descriptions given here rather than restating them.

3.1 Threat Model and Design Goal

The property the architecture is designed to provide is narrow and specific: given a decision record and an explanation that a party claims was produced at decision time, an independent verifier, with no access to the original model or database, can determine whether that record matches what was actually committed on-chain at the time of decision, without trusting the deploying organisation's own logs. The architecture does not attempt to prevent the deploying organisation from choosing not to commit a record at all, nor does it attempt to verify that the model's decision was correct or fair; it verifies only that a committed record has not been altered after the fact. This is a non-repudiation and tamper-evidence guarantee, not a correctness guarantee, and the distinction matters for interpreting the results in Section 6.

3.2 Explanation and Hashing Pipeline

For each decision requiring an audit record, the pipeline performs four steps, shown in the upper two layers of Figure 1. First, the trained model produces a prediction and a probability score. Second, a SHAP TreeExplainer computes an exact, per-feature attribution vector for that specific prediction (Section 4.2). Third, an audit record is assembled combining a record identifier, model identity and version, the decision (prediction, probability, threshold), the ground-truth label where available, the explanation method and its top contributing features, and audit metadata including a UTC creation timestamp and the declared hash algorithm. Fourth, this record is serialized to canonical JSON, with keys sorted, whitespace removed, and UTF-8 encoded, and hashed with SHA-256. Canonical serialization is essential: two logically identical records must hash identically regardless of dictionary insertion order or formatting, or verification would fail on records that were never actually altered.

3.3 On-Chain Commitment: Individual Scheme

The first commitment contract, XAI AuditRegistry, contains one struct for each audit record containing the record ID, transaction ID, 32 bytes of the record SHA-256 hash, a block timestamp and the address of the submitter, in a mapping indexed by the record ID. The registration function will fail if the record with the specified identifier already exists. A view

function allows for checking whether a newly submitted hash matches the one that is stored and returning the result as a Boolean. Hence, any verifier can check or refute a record candidate using just one cheap on-chain read. The cost function for such a scheme is quite clear — one write transaction per audit record, where the gas cost will be dominated by SSTORE calls used to create a new mapping entry, hence, it should be flat for each subsequent record regardless of how many records were registered previously, a hypothesis we test directly in Section 6 using batches of 10 to 5,000 records.

3.4 On-Chain Commitment: Merkle-Batched Scheme

The second contract, `MerkleAuditRegistry`, commits an entire batch of audit records with a single write: a batch identifier, a 32-byte Merkle root [11] computed over the batch's record hashes, a record count, a timestamp, and the submitter. Verification of an individual record's membership in a committed batch is performed by supplying the record's leaf hash together with a Merkle proof, the sibling hashes along the path from that leaf to the root, which the contract recomputes on-chain using sorted-pair keccak256 hashing at each level, comparing the final computed root against the stored one. Sorted-pair hashing, in which the smaller of the two 32-byte values is concatenated first at every level, is used on both the off-chain tree-construction side and the on-chain verification side so that proof verification is independent of leaf ordering within a pair; the off-chain Python implementation and the on-chain Solidity implementation are required to agree on this rule exactly, since any divergence would make every proof fail. This scheme's cost model is fundamentally different from the individual scheme: exactly one write transaction is required per batch regardless of how many records it contains, so the marginal on-chain cost of adding one more record to a not-yet-closed batch is, in the limit, the cost of one additional leaf-hash computation performed off-chain, effectively free from the blockchain's perspective until the batch is closed and committed.

3.5 Off-Chain Storage

In both schemes, the full audit record, including the complete SHAP explanation, is retained off-chain; only the compact commitment (a 32-byte hash, or a 32-byte root plus a short proof) touches the chain. This division is a direct application of the architectural principle, well established in Ethereum engineering practice [9], [10] and adopted in the closest prior XAI-audit proposals [5], that expensive on-chain storage should be reserved for the minimum data needed

to make a tamper-evidence guarantee, with bulk content kept off-chain and referenced rather than replicated on-chain.

4. Implementation

4.1 Dataset and Preprocessing

The fraud-detection model is trained on the IEEE-CIS Fraud Detection dataset's transaction table [4] (590,540 rows, retrieved via the Kaggle competition API), sorted chronologically by transaction time to prevent information leakage from future transactions into the training process. Columns with more than 90% missing values were dropped, retaining 391 features, 14 of which are categorical and handled natively by LightGBM's categorical-split mechanism rather than one-hot encoded. Numeric columns were downcast to reduce memory footprint. The data was split chronologically, not randomly, into a training period of 472,432 transactions (80%) and a held-out test period of 118,108 transactions (20%), and the training period was further split chronologically into a fitting set of 425,188 transactions (90%) and a validation set of 47,244 transactions (10%), so that both threshold selection and final evaluation are tested strictly on transactions that occurred after the data the model and threshold were tuned on.

4.2 Model Training and Threshold Optimisation

A LightGBM [12] gradient-boosted classifier (2,000 estimators, learning rate 0.03, 64 leaves, 0.8 row and column subsampling, early stopping on validation AUC with a 100-round patience) was trained with a class-weighted binary objective; the positive-class weight (`scale_pos_weight` of approximately 27.45) was computed directly from the fitting split's class ratio to counteract the dataset's severe fraud-to-non-fraud imbalance. Training stopped at iteration 579. Rather than using the conventional 0.50 probability threshold, a decision threshold was selected by sweeping candidate thresholds against the validation split and choosing the value maximising validation-set F1; this search selected a threshold of 0.78, subsequently fixed and applied unchanged to the held-out test split for final evaluation and for every explanation and audit record generated downstream. Final test-set performance at this threshold is reported in Section 6.1.

4.3 Explainability Layer

Per-transaction explanations use SHAP's TreeExplainer [2], which computes exact Shapley-value feature attributions for tree-ensemble models in polynomial time rather than relying on model-agnostic sampling approximations, an important property for an audit context, since the explanation is then a deterministic function of the trained model and the input, not a stochastic estimate that could differ between two honest recomputations. For each explained transaction, the ten most influential features by absolute SHAP value are retained in the audit record; a full per-feature SHAP table is additionally retained off-chain for completeness. As a worked example, transaction 3475473, a true-positive fraud case predicted with probability 0.99979, was explained with its top attributions dominated by several of the dataset's anonymised Vesta engineered features (V258, V156, and V149, among others) alongside the recipient and purchaser email-domain fields, illustrating that the explanation layer surfaces genuinely informative, transaction-specific evidence rather than generic global feature rankings.

4.4 Audit Record Schema and Canonical Hashing

Each audit record is assembled as a JSON object with a record identifier of the form AUD-<transactionId>, a record type (AI_FRAUD_DECISION), nested model identity (name and version string), decision fields (prediction, fraud probability, decision threshold), the ground-truth label, an explainability block (method name and the top-feature attribution list), and audit metadata (schema version, UTC ISO-8601 creation timestamp, declared hash algorithm). Canonical serialization sorts all object keys alphabetically and removes all non-essential whitespace before UTF-8 encoding, guaranteeing that byte-identical input always produces byte-identical serialized output regardless of the order in which fields were constructed in memory. SHA-256 is applied to this canonical byte string to produce the record hash used in both on-chain commitment schemes. Section 4.6 gives the corresponding algorithmic specification of how this hash is later re-derived and checked during tamper detection.

4.5 Smart Contracts and Execution Environment

Both contracts are written in Solidity 0.8.34 [15] and compiled with the Solidity optimiser enabled (200 runs) via Hardhat ^3.15.0 [14], using the hardhat-toolbox-mocha-ethers plugin (^3.0.7), ethers.js (^6.17.0), and TypeScript (~6.0.3) on Node.js. Benchmarking was performed against a local, simulated Ethereum-compatible development network (Hardhat's built-in EDR-simulated chain, chain type L1), whose gas-accounting model follows the Ethereum specification [13], exposed over the standard JSON-RPC interface at 127.0.0.1:8545; a Sepolia public-testnet configuration is included in the project for future deployment but was not used for the measurements reported here. Section 7.2 discusses the implications of benchmarking on a local simulated network rather than a public chain; that discussion is not repeated in Section 6, which reports measured values only.

All on-chain interactions were driven from Python 3.13.2 via the web3.py client library (7.16.0), using a single funded local account as the transaction submitter throughout. The fraud-detection and explainability pipeline used pandas 3.0.5, NumPy 2.5.2, scikit-learn 1.9.0, LightGBM 4.7.0, SHAP 0.52.0, and joblib 1.6.0. All software ran on a single Apple Silicon Mac (macOS) workstation; none of the models or on-chain components require GPU acceleration, so results should transfer to any comparable CPU-only environment. Exact CPU and memory specifications were not logged during the original benchmarking runs and are reported here to the extent available; the fixed random seeds and complete dependency versions listed above are given precisely so that the reported figures can be reproduced independent of the specific host machine.

4.6 Tamper-Detection Algorithm

The tamper-detection procedure used throughout Sections 5.4 and 6.5 is specified formally in Algorithm 1. It is deliberately identical in structure for both commitment schemes, differing only in the on-chain call used to check a candidate hash: an equality comparison for the individual scheme, and a Merkle-proof recomputation for the batched scheme.

Algorithm 1: Tamper Detection for a Committed Audit Record

Input: record R (candidate audit record, on-chain or off-chain claim), commitment scheme σ {individual, merkle}

Input: on-chain contract instance C for scheme σ ; for $\sigma = \text{merkle}$, Merkle proof π and batch identifier b

Output: VALID or TAMPERED

1. canonical $\leftarrow$ CanonicalJSON(R) // sort keys, strip whitespace, UTF-8 encode (Sec. 4.4)
 2. $h \leftarrow \text{SHA256}(\text{canonical})$ // recompute the 32-byte record hash
 3. if $\sigma = \text{individual}$ then
 4. recordId $\leftarrow R.\text{recordId}$
 5. valid $\leftarrow C.\text{verifyAuditRecord}(\text{recordId}, h)$ // on-chain equality check (Sec. 3.3)
 6. else if $\sigma = \text{merkle}$ then
 7. leaf $\leftarrow h$
 8. root $\leftarrow \text{leaf}$
 9. for each sibling s in π (leaf to root) do
 10. root $\leftarrow \text{HashPair}(\text{root}, s)$ // sorted-pair keccak256 (Sec. 3.4)
 11. end for
 12. storedRoot $\leftarrow C.\text{getBatchRoot}(b)$
 13. valid $\leftarrow (\text{root} = \text{storedRoot})$ // on-chain root recomputation (Sec. 3.4)
 14. end if
 15. if valid = true then
 16. return VALID
 17. else
 18. return TAMPERED
 19. end if
-

5. Experimental Methodology

5.1 Audit Record Generation

500 audit records were generated from genuine held-out test-set transactions using the trained model and threshold described in Section 4.2, each with a full SHAP TreeExplainer explanation computed as in Section 4.3. These are real decisions and real explanations, not synthetic placeholders constructed only to exercise the benchmark, so the measured costs reflect the actual size and structure of production-representative audit records. Section 6.6 additionally reports a synthetic-hash extension of the benchmark to larger batch sizes, described and justified in that section.

5.2 Benchmark Design

The 500 records were partitioned into cumulative batch sizes of 10, 50, 100, 250, and 500 records. For the individual-commitment scheme, each record in a batch was registered with its own on-chain transaction, as specified in Section 3.3; for the Merkle-batched scheme,

each batch's records were assembled into a Merkle tree off-chain and committed with a single registerBatch transaction, as specified in Section 3.4. Every batch size, under both schemes, was executed for five independent repetitions to characterise variance as well as central tendency, following standard practice for latency benchmarking on a system with non-deterministic scheduling and input/output jitter.

5.3 Metrics and Gas-Reduction Formulation

For each transaction (individual scheme) or each batch (Merkle scheme) the following were recorded: registration latency (client-observed wall-clock time from transaction submission to mined-receipt confirmation), gas consumed (read from the transaction receipt), verification latency (wall-clock time for the corresponding view-function call), and a binary tamper-detection outcome. Per-batch-size summary statistics, namely mean, standard deviation, median, and 95th percentile where applicable, were computed across the repetitions.

Let $G_{ind}(n)$ denote the total gas consumed by the individual-commitment scheme when registering n records, and let $G_{mer}(n)$ denote the total gas consumed by the Merkle-batched scheme when committing the same n records as a single batch. The effective per-record gas cost of the Merkle scheme is defined as:

$$g_{mer}(n) = G_{mer}(n)/n$$

which is the quantity compared directly against the individual scheme's inherently per-record cost, $g_{ind}(n) = G_{ind}(n) / n$. The gas-reduction percentage achieved by adopting the Merkle-batched scheme over the individual scheme at batch size n is defined as:

$$R(n) = [(G_{ind}(n) - G_{mer}(n))/G_{ind}(n)] \times 100\%$$

and the corresponding reduction factor, reported alongside $R(n)$ throughout Section 6, is defined as:

$$F(n) = G_{ind}(n)/G_{mer}(n)$$

Because $G_{ind}(n)$ grows linearly in n (one write transaction per record) while $G_{mer}(n)$ is, to first approximation, constant in n (one write transaction per batch), $R(n)$ approaches 100% and $F(n)$ grows approximately linearly with n as n increases; Section 6.4 and Section 6.6 report the measured values of $R(n)$ and $F(n)$ at each tested batch size against this closed-form expectation. The on-chain transaction-count reduction, reported alongside $R(n)$ and $F(n)$,

follows directly from the same accounting: the individual scheme issues n transactions and the Merkle scheme issues 1, so the transaction-count reduction is $(n - 1)/n \times 100\%$.

5.4 Tamper-Detection Experiment

Two complementary tamper-detection tests were run, formally specified as Algorithm 1 in Section 4.6. On-chain, for every registered record or batch, a single bit of the supplied comparison hash was flipped before calling the contract's verification function, and the outcome (correctly rejected, or incorrectly accepted) was logged; this directly tests the on-chain equality and Merkle-recomputation logic under adversarial input. Off-chain, a full audit record was deep-copied, a single decision field (the fraud probability) was altered from its true stored value, and the canonical-hash procedure of Section 4.4 was reapplied to the altered record; the resulting hash was compared against the originally committed hash to confirm that a single-field change anywhere in the record is sufficient to invalidate the commitment, which is the property the entire architecture depends on.

6. Results

6.1 Fraud-Detection Model Performance

At the validation-optimised decision threshold of 0.78, the model achieved an accuracy of 97.09%, precision of 61.57%, recall of 41.31%, F1 of 0.4945, ROC-AUC of 0.9003, and PR-AUC of 0.5053 on the 118,108-transaction held-out test split. The precision-recall trade-off reflects a deliberate threshold choice that favours precision over recall relative to the default 0.50 cutoff, appropriate for a fraud-flagging context where each positive prediction carries an operational cost (manual review or transaction blocking) that a lower threshold would multiply considerably for comparatively little additional recall, as the full threshold sweep in the underlying experiment data confirms: thresholds below roughly 0.20 recover over 90% of fraud cases but at precision below 15%, whereas 0.78 was the point of maximum harmonic balance on the validation split. This model, not a toy or synthetic classifier, is the one whose real decisions and real SHAP explanations were audited in the blockchain experiments that follow.

6.2 Individual-Commitment Benchmark

Table 1. Individual-commitment scheme: mean $\pm$ SD across 5 repetitions per batch size

Batch Size	Reg. Latency Mean $\pm$ SD (ms)	Verify Latency Mean (ms)	Mean Gas Used	Throughput (records/s)	Tamper Detection
10	6.90 $\pm$ 0.90	2.32	139,298	145.5	100%
50	6.81 $\pm$ 1.57	2.24	139,298	147.5	100%
100	6.26 $\pm$ 0.98	2.07	139,297	159.8	100%
250	6.29 $\pm$ 0.95	2.09	139,297	159.1	100%
500	6.83 $\pm$ 1.77	2.28	139,296	147.1	100%

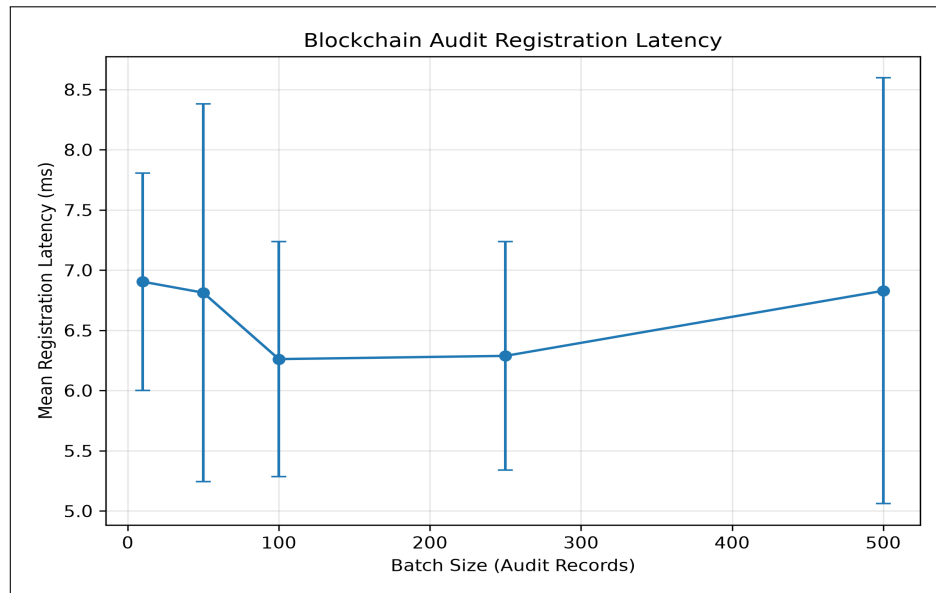


Figure 2. Individual-scheme registration latency, mean $\pm$ standard deviation across five repetitions

Table 1 reports mean and standard deviation across five repetitions at each batch size for the individual-commitment scheme, whose mechanism is specified in Section 3.3. Registration latency was consistently in the 6.3 to 6.9 ms range regardless of batch size, and mean gas per record was effectively constant at approximately 139,297 across all batch sizes, consistent with the flat, batch-size-independent SSTORE cost predicted architecturally. Figure 2 shows this registration latency, with its mean and standard deviation across the five repetitions, plotted against batch size. Verification latency was consistently faster than registration, in the 2.07

to 2.32 ms range, reflecting that verification is a read-only call requiring no transaction mining. Both valid-verification rate and tamper-detection rate were 1.0 (100%) at every batch size across all five repetitions.

6.3 Merkle-Batched Commitment Benchmark

Table 2 reports the corresponding statistics for the Merkle-batched scheme specified in Section 3.4. A single batch registration, regardless of whether it contained 10 or 500 records, required between roughly 6.4 ms and 9.2 ms of client-observed latency and consumed approximately 116,410 to 116,434 gas in total. The effective per-record gas cost $g_{\text{mer}}(n)$, computed as defined in Section 5.3, fell from 11,641 gas per record at a batch of 10 to 232.9 gas per record at a batch of 500. Merkle proof length, the number of sibling hashes a verifier must supply, grew from 4 hashes at a batch of 10 to 9 hashes at a batch of 500, consistent with the logarithmic growth in proof length with batch size reported for general Merkle-tree state proofs [10]. Verification latency was marginally higher than the individual scheme's, at 2.56 to 3.16 ms, reflecting the additional on-chain hash computations needed to walk the proof to the root. Valid-verification and tamper-detection rates were again 1.0 at every batch size.

Table 2. Merkle-batched commitment scheme: mean $\pm$ SD across 5 repetitions per batch size

Batch Size	Batch Reg. Latency $\pm$ SD (ms)	Verify Latency (ms)	Total Batch Gas	Effective Gas/Record	Proof Length	Tamper Detection
10	8.82 $\pm$ 3.55	2.74	116,410	11,641.0	4	100%
50	7.06 $\pm$ 0.18	2.56	116,410	2,328.2	6	100%
100	6.37 $\pm$ 0.15	2.28	116,410	1,164.1	7	100%
250	7.63 $\pm$ 2.49	2.75	116,410	465.6	8	100%
500	9.24 $\pm$ 4.06	3.16	116,434	232.9	9	100%

6.4 Comparative Gas Analysis

Table 3 and Figures 3 to 5 present the direct comparison between the two schemes at matched batch sizes, computed using the $R(n)$ and $F(n)$ definitions of Section 5.3. Gas reduction rose from $R(10) = 91.64\%$ to $R(500) = 99.83\%$, corresponding to a reduction factor of $F(10)$

≈ 12 at the smallest batch tested and $F(500) \approx 598$ at the largest, consistent with, and at the upper end of, gas savings reported for batching and Merkle-proof techniques applied to other smart-contract workloads [9], [10]. Figure 3 shows the effective per-record gas cost of both schemes; Figure 4 shows $R(n)$ as a function of batch size; Figure 5 shows Merkle proof length as a function of batch size. Because the Merkle scheme requires exactly one on-chain transaction per batch rather than one per record, the on-chain transaction count itself falls by 90% at a batch of 10 and by 99.8% at a batch of 500, a reduction with direct implications for network congestion and for wallet or infrastructure nonce management in a production deployment issuing many audit commitments per day.

Table 3. Comparative on-chain gas analysis: individual vs. Merkle-batched commitment (10–500 records)

Batch Size	Individual Total Gas	Merkle Total Gas	Gas Reduction $R(n)$	Reduction Factor $F(n)$	Transaction Reduction
10	1,392,980	116,410	91.64%	11.97×	90.0%
50	6,964,876	116,410	98.33%	59.83×	98.0%
100	13,929,728	116,410	99.16%	119.66×	99.0%
250	34,824,140	116,410	99.67%	299.15×	99.6%
500	69,648,076	116,434	99.83%	598.18×	99.8%

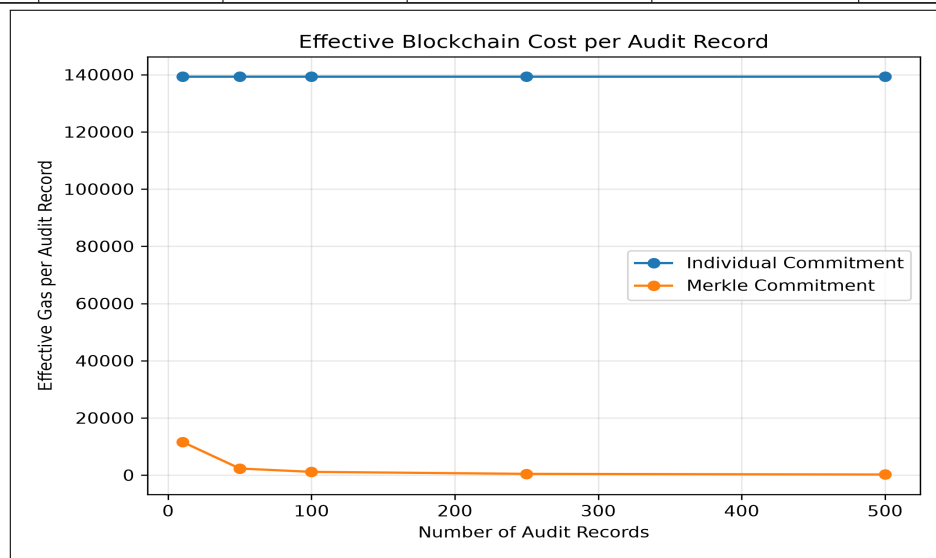


Figure 3. Effective on-chain gas cost per audit record — individual vs. merkle-batched commitment (10–500 records)

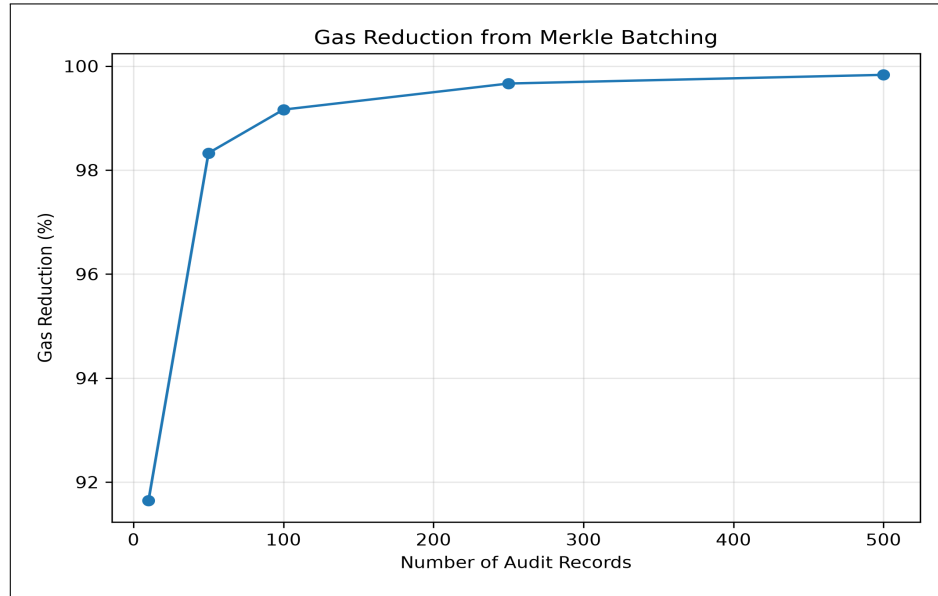


Figure 4. Gas Reduction $R(n)$ achieved by merkle batching as a function of batch size (10–500 records)

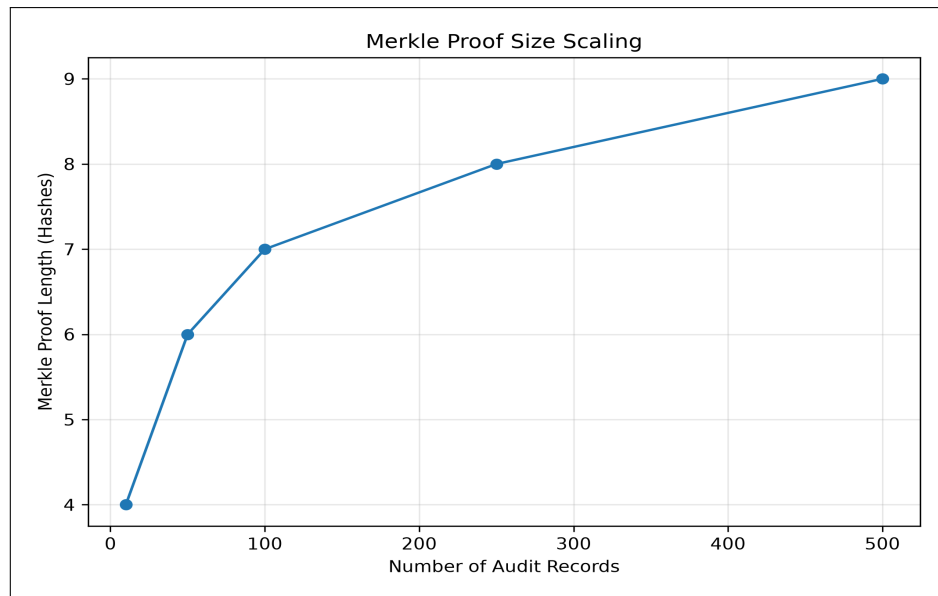


Figure 5. Merkle proof length (sibling hashes required) as a function of batch size (10–500 records)

6.5 Tamper-Detection Results

Across all 5,000 individual on-chain verification trials (500 records times 5 repetitions times both the valid-hash and bit-flipped-hash conditions) and the corresponding Merkle-proof trials, executed following Algorithm 1, tamper detection was correct in every case: a bit-flipped supplied hash was rejected by the individual scheme's equality check with 100% reliability, and a bit-flipped leaf was rejected by the Merkle scheme's proof-recomputation logic with equal reliability. The complementary off-chain experiment altered a real stored decision record's

fraud probability field (from 0.9997908244929554 to 0.499791) and recomputed its canonical SHA-256 hash: the altered record's hash differed from the originally committed hash in every byte position examined, consistent with the expected avalanche property of SHA-256, and the verification procedure correctly reported the record as TAMPERED rather than VALID. Together, these results confirm the architecture's core non-repudiation property empirically rather than only by cryptographic assumption: any post-hoc alteration of a committed decision or explanation, however small, is detectable by any party holding only the originally published commitment.

6.6 Extended Scalability Analysis

The 500-sample size used in Sections 6.2 - 6.5 is based on the expense of producing true SHAP explanations for out-of-sample fraud detections. Every extra audit record will need full model inference and TreeExplainer calculation for each of them. Thus, 500 records were selected as a pilot scale which would be sufficient for modelling cost with five-repeat statistical significance while making the explanation process possible. Given that the on-chain cost of commitment scheme depends only on the quantity and byte length of the committed hash values and not on the meaning behind them, the blockchain-side cost model derived in Sections 6.2 - 6.4 can be verified independently from any more SHAP calculations by extending the benchmark set with more structurally similar 32-byte commitment hashes.

In order to evaluate whether the patterns seen until 500 records continue when scaling to higher production-related volumes, the benchmark was scaled to batches containing 1,000, 2,500, and 5,000 records. This involved batches made up of the initial 500 audit records along with an additional 500, 2,000, and 4,500 commitment hashes which were produced synthetically. The synthetic hashes were produced by hashing random seed strings using SHA-256 and assigning them transaction IDs from another numeric range (above 9,000,000). This ensures that no synthetic record can be considered as a legitimate transaction. Two experiments were performed at each of the scaled-up batch sizes.

Table 4. Extended scalability analysis: individual vs. Merkle-batched commitment (1,000–5,000 records, synthetic commitment hashes)

Batch Size	Individual Total Gas	Merkle Total Gas	Gas Reduction R(n)	Reduction Factor F(n)	Merkle Proof Length
1,000	139,308,284	116,446	99.9164%	1,196.33×	10
2,500	348,289,112	116,434	99.9666%	2,991.30×	12
5,000	696,590,212	116,446	99.9833%	5,982.09×	13

The individual scheme's per-record gas cost remained flat across the extended range, at 139,308 gas at $n = 1,000$ and 139,318 gas at $n = 5,000$, a variation of under 0.01% attributable to the marginally longer record identifiers used for synthetic records. The Merkle scheme's effective per-record gas cost continued to fall as $g_{\text{mer}}(n) \approx 116,440/n$, reaching 116.45 gas per record at $n = 1,000$ and 23.29 gas per record at $n = 5,000$, while its Merkle proof length grew from 10 hashes at $n = 1,000$ to 13 hashes at $n = 5,000$, consistent with the logarithmic growth already observed up to $n = 500$. Applying the $R(n)$ and $F(n)$ definitions of Section 5.3 to these extended results gives $R(5,000) = 99.98\%$ and $F(5,000) \approx 5,982$, roughly a tenfold increase in reduction factor over $F(500) \approx 598$.

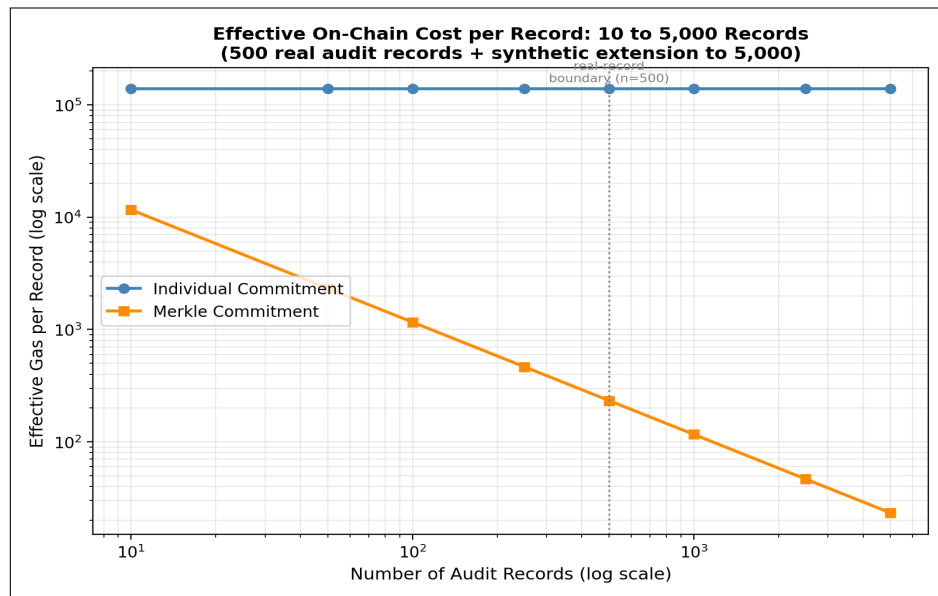


Figure 6. Effective on-chain cost per record, 10 to 5,000 records (log–log), real records to $n = 500$ and synthetic extension beyond

Figure 6 plots the effective per-record gas cost of both schemes on logarithmic axes

across the full 10-to-5,000-record range, showing the individual scheme's flat line and the Merkle scheme's linear decline continuing smoothly across the boundary between real and synthetic records at $n = 500$. Both schemes retained a 100% valid-verification rate and a 100% tamper-detection rate at every extended batch size, using the same procedure as Section 6.5.

7. Discussion

7.1 Interpreting the Cost Model

The two schemes specified in Sections 3.3 and 3.4 are not competing alternatives so much as complementary tools for different points in the decision-volume and latency-tolerance space. The individual scheme is well suited to lower-volume, higher-stakes decisions, such as a declined mortgage application or a blocked six-figure transfer, where the cost per commitment is small relative to the stakes and auditors will want to check individual decisions in isolation without needing a proof object. The Merkle-batched scheme is well suited to high-volume, lower-per-decision-stakes settings, such as a card-fraud-detection pipeline issuing thousands of automated declines per day, where committing every decision individually would be gas-prohibitive at scale but a delay of minutes to hours until a batch closes is operationally acceptable. This distinction follows directly from the closed-form cost model of Section 5.3: $R(n)$ and $F(n)$ both increase with n , so the case for batching strengthens monotonically as decision volume grows, a trend confirmed empirically up to $n = 5,000$ in Section 6.6.

The practical design question this raises for a real deployment is batching cadence: how large a batch, and how long a delay before closing it, an operator is willing to accept in exchange for the measured gas savings. The results in Table 3 and Table 4 suggest diminishing marginal returns are already substantial by a batch of 100 to 250 records ($R(n)$ of 99.16% to 99.67%), so an operator need not accumulate very large batches, or tolerate very long delays, to capture the great majority of the available saving; batching to 5,000 buys a further, still meaningful increment ($R(n)$ rising from 99.67% to 99.98%, $F(n)$ rising roughly tenfold from $F(500)$ to $F(5,000)$) at the cost of a longer proof and a longer commitment delay.

7.2 Latency, Throughput, and Network Realism

All latency figures reported in Section 6 were measured against a local, simulated Ethereum-compatible development node with a single submitting account issuing transactions

sequentially (Section 4.5). This isolates the protocol- and method-call-level overhead of each scheme cleanly, which is the comparison this paper is designed to make, but it necessarily excludes factors that dominate wall-clock experience on a public network: peer-to-peer transaction propagation, block-time confirmation delay (of the order of seconds on Ethereum mainnet and public testnets, faster on many rollups), mempool congestion, and gas-price volatility. The millisecond-scale registration latencies reported here should therefore be read as a lower bound characterising the two schemes' relative overhead, not as a prediction of end-to-end confirmation time on a production network; a deployment on a public chain or a rollup would need its own latency characterisation under realistic network conditions, though the relative gas-cost comparison between the two schemes, which depends on EVM execution and storage cost rather than network propagation, should transfer directly. This is the only point in the paper at which the implications of the local-network choice are discussed; Sections 4.5 and 6 report the choice and the resulting measurements respectively without re-interpreting them.

7.3 Trust Model and Limitations

Several limitations bound the claims made in this paper. First, both example contracts expose their registration functions to any caller; a production deployment auditing a specific institution's decisions would need access control restricting registration to an authorised submitter address or role, to prevent spoofed commitments falsely claiming to originate from that institution. This is a straightforward extension (an owner-restricted modifier or a role-based access-control library) but is not present in the benchmarked contracts and should not be assumed. Second, the tamper-detection experiments, while conclusive for the specific alterations tested (single-bit hash flips on-chain, a single decision-field change off-chain), do not constitute an exhaustive adversarial fuzzing campaign across every field and structural variation an adversary might attempt; the SHA-256 avalanche property gives strong theoretical grounds to expect equally reliable detection for any other single- or multi-field alteration, but this was not separately measured for every field. Third, the architecture commits a declared `model_version` string rather than a cryptographic hash of the model artifact itself, so it does not, on its own, prove that the explanation was produced by a specific model binary; it proves only that the record naming that version has not been altered since commitment. Extending the

schema to include a model hash would close this gap and is a natural direction for future work. Fourth, all benchmarking used a single funded account submitting transactions sequentially; concurrency effects from multiple simultaneous submitters, which would be expected in a real multi-service deployment, were not characterised. Fifth, the benchmarks were run on a local simulated network rather than a public testnet, mainnet, or rollup, for the reasons and with the caveats discussed in Section 7.2. Sixth, the extended-scale results in Section 6.6 validate the blockchain-side cost model, gas, proof length, and transaction count, using synthetic commitment hashes beyond $n = 500$; they do not demonstrate that 5,000 genuine SHAP explanations were generated, since the on-chain cost of a commitment is independent of the semantic origin of the hash but the cost of producing a genuine explanation is not, and that latter cost was characterised only up to the 500 real records used in Sections 6.1 to 6.5.

8. Conclusion

This research study has presented a two-tier blockchain architecture for verifiable explainable-AI audit trails, implemented end to end on a real fraud-detection pipeline: a LightGBM classifier trained and threshold-tuned on the IEEE-CIS Fraud Detection dataset, a SHAP TreeExplainer layer producing genuine per-transaction feature attributions, a canonical hashing procedure guaranteeing deterministic commitment, and two complementary Solidity smart contracts committing those hashes individually or in Merkle-batched form. Benchmarked on 500 real audit records across batch sizes from 10 to 500, repeated five times each, and further validated on synthetic but structurally identical commitment hashes up to 5,000 records, the individual scheme provides immediate, flat-cost, single-hash-comparison verification at approximately 139,300 gas per record across the full range, while the Merkle-batched scheme reduces total on-chain gas from 91.6% at 10 records to 99.98% at 5,000 records, reaching a reduction factor of nearly 5,982-fold, at the cost of a longer inclusion proof and a delayed batch-close commitment point. Both schemes detected 100% of tampering attempts across thousands of on-chain and off-chain trials at every tested scale, confirming the architecture's core non-repudiation guarantee empirically. Future work includes benchmarking on a public testnet or rollup to characterise realistic network latency; adding access control and model-artifact hashing to the commitment schema; extending adversarial tamper testing across all record fields; generating genuine SHAP-derived audit records beyond the 500-record pilot

scale used here, to complement the synthetic-hash extension reported in Section 6.6 with a fully real-data validation at the same volumes; and evaluating the architecture against additional explanation methods beyond SHAP TreeExplainer.

References

- [1] Goodman, Bryce, and Seth Flaxman. “European Union Regulations on Algorithmic Decision-Making and a “Right to Explanation”.” *AI Magazine* 2017, vol. 38, no. 3, 50-57.
- [2] Lundberg, Scott M., and Su-In Lee. “A Unified Approach to Interpreting Model Predictions.” *Advances in Neural Information Processing Systems* 2017, vol. 30, 1-10.
- [3] Nakamoto, Satoshi, and A. Bitcoin. “A Peer-to-Peer Electronic Cash System.” *Bitcoin*.—URL: <https://bitcoin.org/bitcoin.pdf> 4, no. 2 (2008): 15.
- [4] Vesta Corporation and IEEE Computational Intelligence Society, “IEEE-CIS Fraud Detection,” *Kaggle Competition Dataset*, 2019. [Online]. Available: <https://www.kaggle.com/c/ieee-fraud-detection>
- [5] Malhotra, Diksha, Shubham Srivastava, Poonam Saini, and Awadhesh Kumar Singh. “Blockchain Based Audit Trailing of XAI Decisions: Storing on IPFS and Ethereum Blockchain.” *International Conference on COMMunication Systems & NETWORKS (COMSNETS)*, IEEE, 2021, 1-5.
- [6] Kulothungan, Vikram. “Using Blockchain Ledgers to Record AI Decisions in IoT.” *IoT* 2025, vol. 6, no. 3, 37.
- [7] Sachan, Swati, and Xi Liu. “Blockchain-based Auditing of Legal Decisions Supported by Explainable AI and Generative AI Tools.” *Engineering Applications of Artificial Intelligence* 2024, vol. 129, 107666.
- [8] Akther, Afroja, Ayesha Arobee, Abdullah Al Adnan, Omum Auyon, A. S. M. Islam, and Farhad Akter. “Blockchain As a Platform for Artificial Intelligence (AI) Transparency.” *arXiv preprint arXiv:2503.08699* (2025).
- [9] Masla, Nitima, Vaibhav Vyas, Jyoti Gautam, Rabindra Nath Shaw, and Ankush Ghosh. “Reduction in Gas Cost for Blockchain Enabled Smart Contract.” *4th International Conference on Computing, Power and Communication Technologies (GUCON)*, IEEE,

2021, 1-6.

- [10] Mizrahi, Avi, Noam Koren, Ori Rottenstreich, and Yuval Cassuto. “Traffic-aware Merkle Trees for Shortening Blockchain Transaction Proofs.” *IEEE/ACM Transactions on Networking* 2024, vol. 32, no. 6, 5326-5340.
- [11] Merkle, Ralph C. “A Digital Signature Based on a Conventional Encryption Function.” In *Conference on the Theory and Application of Cryptographic Techniques*, Springer Berlin Heidelberg, 1987, 369-378.
- [12] Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. “Lightgbm: A Highly Efficient Gradient Boosting Decision Tree.” *Advances in Neural Information Processing Systems* 2017, vol. 30, 1-9.
- [13] Wood, Gavin. “Ethereum: A Secure Decentralised Generalised Transaction Ledger.” *Ethereum Project Yellow Paper* 2014, vol. 151, no. 2014, 1-32.
- [14] Nomic Foundation, “Hardhat Documentation,” 2025. [Online]. Available: <https://hardhat.org/docs>
- [15] Ethereum Foundation, “Solidity Documentation, v0.8.34,” 2025. [Online]. Available: <https://docs.soliditylang.org>