International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(9s) 2026 pp. 231-244
https://doi.org/10.70917/ijcisim-2026-3437
Article

AgriChain: Design, Implementation and Empirical Evaluation of an Ethereum Smart-Contract Framework for Agricultural Supply-Chain Traceability

Rajiv Ghai[1*], Anil Kumar Bisht[2]

[1*]Department of Computer Science and Information Technology, MJP Rohilkhand University, Bareilly, India (Corresponding author) Email: rajivghai7@mjprudor.ac.in, ORCID: 0009-0009-8022-8892
[2]Department of Computer Science and Information Technology, MJP Rohilkhand University, Bareilly, India Email: anil.bisht@mjpru.ac.in, ORCID: 0000-0002-5629-8809

Abstract: Agricultural supply chains remain vulnerable to provenance fraud, cold-chain opacity and inefficient recall investigation because batch records are typically fragmented across paper documents and siloed enterprise databases that are neither tamper-evident nor independently verifiable. This paper presents AgriChain, an Ethereum smart-contract framework that records batch initialisation, supply-chain events, ownership transfers, batch splitting, batch merging and IoT-style oracle sensor readings on a public blockchain, paired with an off-chain SQLite indexer and a Flask dashboard for fast querying. The system was deployed on the Ethereum Sepolia test network and its behaviour is evaluated using a fully block-traceable primary experiment (14 transactions spanning the complete lifecycle of a rice batch and a two-batch corn merge) together with the cumulative on-chain history recorded by the deployment (9 batches, 16 events, 3 merges and 8 oracle submissions (all 8 verified). Independent recompilation of the contract confirms a deployed bytecode size of 19,163 bytes, comfortably within Ethereum's 24,576-byte EIP-170 limit. Measured gas consumption is reported per function (from 38,436 gas for an ownership transfer to 691,659 gas for a mass-balance-checked split). Critically, cross-referencing the on-chain evidence against the local database uncovered a genuine synchronisation defect: confirmed on-chain split operations are silently absent from the off-chain indexer and all exports derived from it, a fault we root-caused to a nested-SQLite-connection locking conflict and confirmed by direct reproduction. We report this defect, its mechanism and its practical implications for any blockchain traceability system that pairs an authoritative ledger with a local query index. The paper concludes with an honest account of the system's current limitations including an open (non-allow-listed) oracle-submission function and an unused batch-recall state and outlines concrete future work.

Keywords: Agricultural supply chain; Blockchain; Ethereum; Food safety; IoT oracle; Solidity; Smart contracts; Traceability

1. Introduction

Roughly one-third of all food produced for human consumption is lost or wasted every year, an estimated 1.3 billion tonnes and in developing countries about 40% of this loss occurs at the post-harvest and processing stages of the supply chain, before food ever reaches a retail shelf [1]. A substantial share of this loss is attributable not to a lack of food but to a lack of information: when a batch of produce changes hands between farmers, aggregators, cold-storage operators, distributors and retailers, the records describing its origin, handling conditions and custody are typically kept in incompatible paper logs or siloed enterprise systems. This fragmentation makes it difficult to verify provenance claims, slows recall investigations when contamination is suspected and creates opportunities for mislabelling and fraud [2].

Blockchain technology has been proposed as a structural response to this problem. Because a blockchain ledger is replicated, append-only and secured by cryptographic hashing and a decentralised consensus mechanism, a batch record written to it cannot be silently altered by any single participant and every downstream party can verify its

history independently rather than trusting a counterparty's database [3], [4]. Ethereum in particular offers a public, permissionless, Turing-complete execution environment in which the traceability logic itself not just the data is enforced identically for every participant by the Ethereum Virtual Machine (EVM) [5], [6]. A growing body of research has proposed blockchain-based traceability architectures for agriculture and food, ranging from RFID-integrated systems for specific national supply chains [7] to HACCP-aligned frameworks that combine blockchain with Internet-of-Things (IoT) sensing [8] and the topic has matured into systematic reviews that catalogue dozens of proposed designs [9], [10].

However, a recurring gap in this literature is the distance between a proposed design and a fully block-verifiable empirical deployment. Many published systems are validated through simulation, a private/permissioned test network, or a small number of illustrative transactions without a complete, independently checkable transaction trail; fewer still report the practical software-engineering difficulties encountered when a blockchain ledger is paired with an off-chain index for fast querying a near-universal architectural pattern in real deployments, since querying a smart contract's full history directly is often impractical for dashboard-style applications. This paper addresses both gaps directly. We present AgriChain, a Solidity smart-contract system for agricultural batch traceability, deployed on Ethereum's public Sepolia test network and we report its evaluation using only evidence that is independently reproducible: on-chain transaction hashes, block numbers and gas measurements, cross-checked against the system's own SQLite index and JSON export. This cross-checking process itself surfaced a genuine defect a class of bug relevant to any system with this architecture which we analyse in detail rather than omit.

The contributions of this paper are: (i) the design and Solidity implementation of a six-operation agricultural batch-traceability contract with a keccak256 hash-chained event log and a confidence-scored oracle-submission mechanism; (ii) deployment on Ethereum Sepolia together with a fully block-traceable primary experiment and the system's cumulative on-chain history, reported with real transaction hashes, block numbers and gas costs; (iii) independent recompilation of the contract to verify its deployed bytecode size against the EIP-170 limit [11], rather than relying on an unverified claim; (iv) the identification, root-cause analysis and empirical reproduction of a synchronisation defect between the authoritative on-chain ledger and the off-chain SQLite indexer, together with its generalisable implications for dual-persistence blockchain architectures; and (v) a candid discussion of the system's current limitations, including an open oracle-submission function and an unused contract state, with concrete directions for future work.

The remainder of the paper is organised as follows. Section 2 reviews related work in blockchain-based agricultural traceability, smart-contract platforms and blockchain-oracle integration. Section 3 describes the system design. Section 4 details the implementation and deployment. Section 5 reports empirical results, including the discovered indexer defect. Section 6 discusses implications and limitations and Section 7 concludes.

2. Related Work

2.1 Blockchain for Agricultural and Food Supply-Chain Traceability

Traceability the ability to retrieve the history, application and location of a product through recorded identification [12] has long been recognised as central to food safety and quality management, predating blockchain by at least two decades [12]. Tian [7] proposed one of the earliest blockchain-based agricultural traceability designs, combining RFID tagging with blockchain storage for a Chinese agri-food supply chain and subsequently extended the approach to align explicitly with Hazard Analysis and Critical Control Points (HACCP) methodology, integrating IoT sensing for real-time condition monitoring [8]. Kshetri [13] analysed blockchain's economic rationale for supply-chain management more broadly, arguing that its principal value lies in deterring falsification by any single participant, including vertically integrated actors who might otherwise control both the product and its records. Wang, Han and Beynon-Davies [9] conducted a systematic literature review of blockchain for supply chains and identified traceability and provenance verification as the most frequently cited application domain, while also noting that a large share of the reviewed studies remained conceptual or simulation-based rather than empirically deployed. Kamble, Gunasekaran and Sharma [14] modelled the factors influencing blockchain-enabled traceability adoption specifically within agricultural supply chains and Sunny, Undralla and Pillai [15] presented an overview of blockchain-based traceability with a working demonstration, similarly emphasising the practical implementation choices that separate a conceptual model from a functioning system. Most recently, Sharma et al. [16] proposed a secure, decentralised agri-food supply chain framework using private Ethereum 2.0 and IPFS, employing a unified smart contract architecture to reduce complexity and gas consumption, a design philosophy consistent with the single-contract approach adopted in the present work. These studies collectively establish both the motivation for blockchain-based agricultural traceability

and a recurring observation: the gap between architectural proposals and fully block-verifiable, empirically reported deployments remains only partially closed.

2.2 Blockchain and Smart-Contract Platforms

The blockchain concept a chain of cryptographically linked, timestamped data blocks maintained by a decentralised peer-to-peer network without a trusted third party originates with Nakamoto's proposal for Bitcoin [4]. Ethereum extended this idea from a ledger of currency transactions to a general-purpose, Turing-complete execution environment in which arbitrary programs (smart contracts) run identically on every node [5], [6]. Because deployed contract bytecode is replicated across the entire network, Ethereum imposes a hard ceiling on deployed contract size, introduced as EIP-170 and fixed at 24,576 bytes, specifically to bound the disk-read, pre-processing and Merkle-proof costs that a call to an oversized contract would impose on every node [11]. In September 2022, Ethereum transitioned its consensus mechanism from Proof-of-Work to Proof-of-Stake in an upgrade known as the Merge, which the Ethereum Foundation reports reduced the network's energy consumption by approximately 99.95% [17]; the Sepolia test network used in this study runs the same post-Merge Proof-of-Stake consensus as Ethereum Mainnet, making it a representative environment for evaluating smart-contract behaviour and gas costs ahead of a production deployment.

2.3 IoT and Oracle Integration for Blockchain Traceability

A blockchain's consensus mechanism can only validate data that originates on-chain; it has no native mechanism for verifying facts about the physical world, such as a sensor's temperature reading. This limitation is widely known as the oracle problem [18]. Reyna et al. [19] surveyed the challenges of integrating blockchain with IoT more broadly, noting that resource-constrained sensor devices and the trustworthiness of the bridge between physical sensors and on-chain state are open problems rather than solved ones. Ellis, Juels and Nazarov [20] proposed Chainlink, a decentralised oracle network that aggregates readings from multiple independent node operators and an off-chain reputation system to mitigate single-source manipulation, in contrast to a design (such as the one evaluated here) in which any address may submit a reading directly. Caldarelli [18] provides a systematic framing of the oracle problem for blockchain applications generally and argues that many deployed systems under-report or do not explicitly address how oracle data is authenticated before being trusted on-chain an observation directly relevant to the oracle-submission design discussed in Section 6 of this paper.

2.4 Positioning of the Present Work

Relative to the studies surveyed above, this paper's distinguishing characteristic is not a novel consensus mechanism or a new cryptographic primitive the mass-balance check used during batch splitting, for example, is a single Solidity require() statement, a standard and expected pattern rather than a new invariant-enforcement technique. Instead, the contribution is methodological rigour in reporting: every transaction referenced in Section 5 is anchored to a real Sepolia block number, gas measurement and transaction hash, the deployed contract's bytecode size is independently re-derived rather than asserted and unlike the great majority of reported deployments, which describe only successful outcomes this paper documents a genuine implementation defect discovered during evaluation, together with its root cause and a reproduced confirmation. We are not aware of prior agricultural-blockchain traceability papers that report this class of on-chain/off-chain consistency failure with an accompanying root-cause reproduction; we consider this a useful, generalisable data point for practitioners designing similar dual-persistence systems.

3. System Design

3.1 Architectural Overview

AgriChain is organised into three layers, shown in Fig. 1. The on-chain layer is a single Solidity contract, AgriChain.sol, deployed on the Ethereum Sepolia test network, which acts as the authoritative, tamper-evident record of every batch, event, transfer, split, merge and oracle submission. The client/off-chain layer, eth_client.py, is a Python command-line application built on web3.py that signs and submits transactions to the contract and maintains a local SQLite database (agrichain_eth.db) mirroring on-chain state for fast querying. The presentation layer, eth_web_app.py, is a Flask web application that reads from the SQLite mirror to render a dashboard and batch-detail pages and can also submit new transactions through the same client class. This is a standard and widely used pattern for blockchain applications: because iterating a smart contract's full historical state directly is often impractical for interactive use, an off-chain index is maintained alongside the ledger. As Section 5.4 shows, the correctness of this mirror is not automatic and must be engineered carefully.

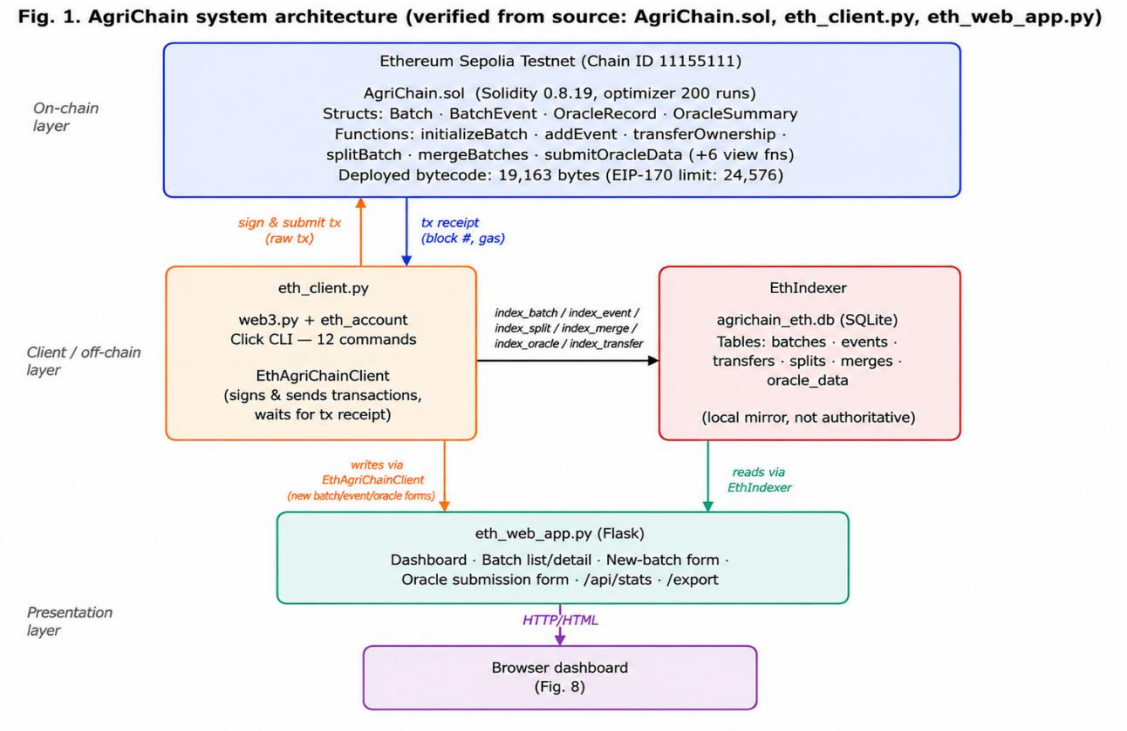

Figure 1.AgriChain three-layer architecture, reconstructed directly from AgriChain.sol, eth_client.py and eth_web_app.py.

3.2 On-Chain Data Model

The contract defines four structs. A Batch holds a batch identifier, product name, origin, initial and current quantity, unit, owner address, a status drawn from the enumeration {Active, Split, Merged, Recalled}, a creation timestamp, an optional parent-batch identifier, a bytes32 integrity hash and an existence flag. A BatchEvent records an event type, location, free-text description, temperature and humidity readings, the submitting address, a timestamp, a sequence number and an event hash. An OracleRecord holds a data type, value, unit, a 0-100 confidence score, a source identifier, the submitting (validator) address, a boolean verified flag and a timestamp. An OracleSummary accumulates a running total value, total confidence and count per data type per batch, so that a consensus average can be computed off-chain without on-chain floating-point arithmetic. We note as a design observation that the BatchStatus enumeration defines a fourth state, Recalled, which is never assigned by any function in the current contract; it appears to be reserved for a future recall/quarantine workflow that has not yet been implemented and we return to this in Section 6.3.

3.3 Contract Functions and Access Control

Six external state-changing functions implement the traceability workflow, gated by three modifiers: onlyExists (requires the batch to be present), onlyActive (requires existence and Active status) and onlyOwner (requires existence and that the caller's address matches the batch's recorded owner).

initializeBatch(batchId, productName, origin, quantity, unit) creates a new batch, provided no batch with the same identifier already exists, the quantity is strictly positive and the identifier is at most 32 bytes. It computes an initial integrity hash as keccak256(batchId || quantity || block.timestamp), sets status to Active and increments a global totalBatches counter.

addEvent(batchId, eventType, location, description, temperature, humidity), gated by onlyActive and onlyOwner, appends a new event. Rather than storing a separate previous-hash field inside each event record, the contract chains events by folding forward a single mutable field on the batch: the new event's hash is computed as keccak256(currentIntegrityHash || eventType || location || block.timestamp) and the batch's integrityHash is then overwritten with this value. Because each new hash is a function of the immediately preceding hash, altering any past event, were it possible, would be detectable by any party who recomputes the chain from the genesis hash forward

and compares it against the value currently stored on-chain, the same principle underlying a blockchain's own block-header chain, applied here at the application level.

transferOwnership(batchId, newOwner, note), gated by onlyActive and onlyOwner, reassigns the owner field and extends the integrity-hash chain to include the transfer. splitBatch(parentId, childAId, childBId, qtyA, qtyB, reason), gated by onlyActive and onlyOwner, requires that qtyA + qtyB exactly equal the parent's current quantity and that both quantities be strictly positive; on success it creates two new Active child batches whose combined quantity conserves the parent's, sets the parent's status to Split and its current quantity to zero and records the parent–child relationship. mergeBatches(sourceAId, sourceBId, mergedId, reason) requires both sources to be Active and owned by the caller and requires their product name and unit to match (compared via keccak256 hash equality on the strings); on success it creates one new Active batch holding the summed quantity and marks both sources Merged. submitOracleData(batchId, dataType, value, unit, confidence, sourceId), gated only by onlyExists, records a sensor-style reading and marks it verified if confidence is at least 80; notably, this function carries no ownership or allow-list restriction, so any Ethereum address may submit a reading against any existing batch the recorded validator address establishes who submitted a reading, but not that the submitter was an authorised sensor operator. We discuss the implications of this design choice in Section 6.3 rather than treating it as an implicit strength.

3.4 Client, Indexer and Dashboard

eth_client.py exposes twelve Click-based CLI commands (init-batch, add-event, transfer, split, merge, oracle, get-batch, get-events, search, stats, export and balance) built around an EthAgriChainClient class that wraps web3.py: each state-changing command builds and signs a transaction, submits it via eth_sendRawTransaction and blocks on wait_for_transaction_receipt (600-second timeout) before reporting the confirmed block number and gas used. Every successful transaction is also written to a local SQLite database, agrichain_eth.db, through an EthIndexer class with six tables - batches, events, transfers, splits, merges and oracle_data each carrying the originating transaction hash and block number alongside the application data, so that every locally indexed row remains traceable back to its on-chain origin. eth_web_app.py is a Flask application that reads exclusively from this SQLite mirror for its dashboard, batch-list, batch-detail and statistics pages (routes /, /batches, /batches/<id>, /stats, /export and a small JSON API) and can also submit new batches, events and oracle readings by invoking the same client class from its POST handlers.

4. Implementation and Deployment

AgriChain.sol is written in Solidity 0.8.19 and, per the compiler settings recorded in the contract source, is intended to be compiled with the optimizer enabled at 200 runs. The system was deployed to the Ethereum Sepolia test network (Chain ID 11155111), a public, Proof-of-Stake test network that mirrors Ethereum Mainnet's execution semantics, using an externally owned account visible throughout the transaction evidence in Section 5 as 0x28e709d3a43Fc38D11De85174aeb39583DcFAd85. The deployed contract address, 0x4a03Fdc7e03909232a9c75A2d0973Ca3e16a25e7, is shown in the connection banner of the system's own dashboard (Fig. 9).

4.1 Independent Bytecode-Size Verification

Rather than relying on an unverified size claim, we independently recompiled the exact contract source using solc 0.8.19 with the optimizer enabled at 200 runs, matching the settings documented in the source file. The resulting creation bytecode is 19,195 bytes and the deployed (runtime) bytecode is 19,163 bytes comfortably within the 24,576-byte EIP-170 ceiling [11], with 5,413 bytes (approximately 5.3 KiB) of headroom remaining. This is worth stating plainly because oversized-contract remediation is a common discussion point in the smart-contract literature: for the single-contract design evaluated here, no bytecode-reduction engineering (splitting into multiple contracts, proxy patterns, or library extraction) is necessary at the current feature set and none was applied.

4.2 Deployment Timeline

The system's on-chain history, reconstructed from block numbers and timestamps across the evidence set, spans two distinct periods. An initial deployment/testing phase between 18 and 20 March 2026 (Sepolia blocks 10,466,707–10,482,361) established six product batches Organic Wheat, Organic Kiwi, two Alphonso Mango batches (later merged), Organic Grapes and Organic Apple along with associated events, a transfer and a merge. A second phase on 22 May 2026 (Sepolia blocks 10,896,001–10,896,291) executed the fully traced primary experiment described in Section 5.1: the complete lifecycle of an Organic Rice batch (initialisation, five events, an ownership transfer, a mass-balance-checked split and three oracle submissions) together with a two-batch Organic Corn merge. This two-phase

history is consistent with iterative development and testing rather than a single scripted run and we report both phases separately in Section 5 to avoid conflating them.

5. Results

5.1 Primary Traced Experiment

Table 1 reports the complete, fully traced 14-transaction lifecycle of batch RICE_001 together with the CORN_001/CORN_002 merge, each row corresponding directly to a screen-captured, confirmed transaction (Figs. 2–8). Every value in this table the operation, block number, gas used and transaction-hash prefix is transcribed directly from these captures; none are estimated or interpolated. Figs. 2 and 3 capture the batch initialisation and the first three supply-chain events (harvest, quality check, and transport) as confirmed on Sepolia. Figs. 4 and 5 show the subsequent cold-storage and dispatch events and the ownership transfer respectively. Fig. 7 shows the three oracle submissions against the two child batches. Fig. 8 shows the CORN_001 and CORN_002 initialisation and the subsequent merge into CORN_MERGED.

Table 1.Primary traced experiment: 14 confirmed Sepolia transactions (* = mass-balance invariant 600 + 400 = 1,000 verified by the EVM before any state change).

Operation	Batch ID	Qty (kg)	Block #	Gas used	Tx hash (prefix)
init-batch	RICE_001	1,000	10,896,001	321,039	1ab9f01f87f35b5e…
add-event [harvest]	RICE_001	—	10,896,023	288,534	4706e24e4bf20cc4…
add-event [quality_check]	RICE_001	—	10,896,027	274,393	8b403ba3b937d598…
add-event [transport]	RICE_001	—	10,896,037	274,342	506b44cd4ffadaba…
add-event [cold_storage]	RICE_001	—	10,896,070	274,441	285a31250d237960…
add-event [dispatch]	RICE_001	—	10,896,073	274,417	9ce65267545ed924…
transferOwnership	RICE_001	—	10,896,251	38,436	d09c66fb7b7f461b…
splitBatch *	RICE_001 → DELHI + MUMBAI	600+400	10,896,257	691,659	28c42ce957414720…
submitOracleData [temp, 95%]	RICE_001_DELHI	—	10,896,279	388,335	602a236b9c8b9d55…
submitOracleData [humidity, 90%]	RICE_001_DELHI	—	10,896,281	323,429	99f822738ce87e11…
submitOracleData [temp, 92%]	RICE_001_MUMBAI	—	10,896,282	354,147	b5a66a1db7a8a2e6…
init-batch	CORN_001	300	10,896,287	286,779	957733210e37f9e8…
init-batch	CORN_002	200	10,896,290	286,767	a66ad512506419fc…
mergeBatches	CORN_001+002 → CORN_MERGED	500	10,896,291	369,154	8f0129d96285020b…

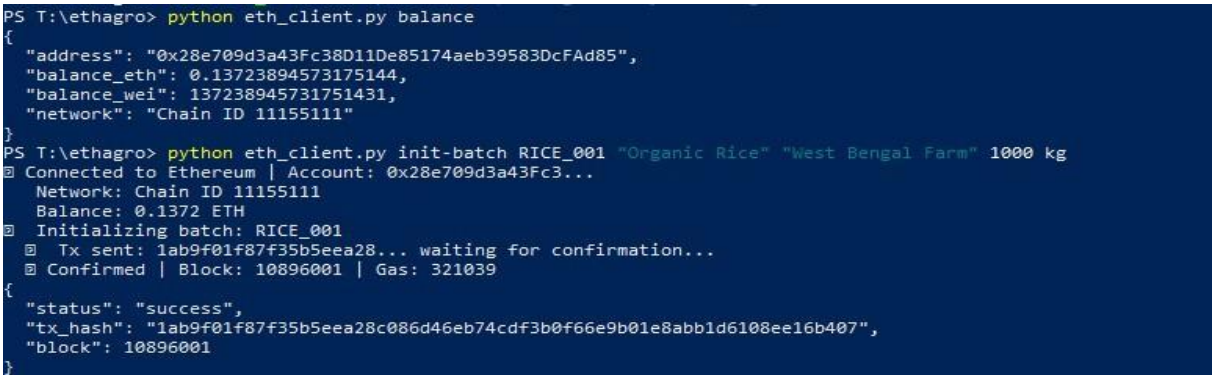

Figure 2.Wallet balance check and RICE_001 initialisation, confirmed Block 10,896,001.

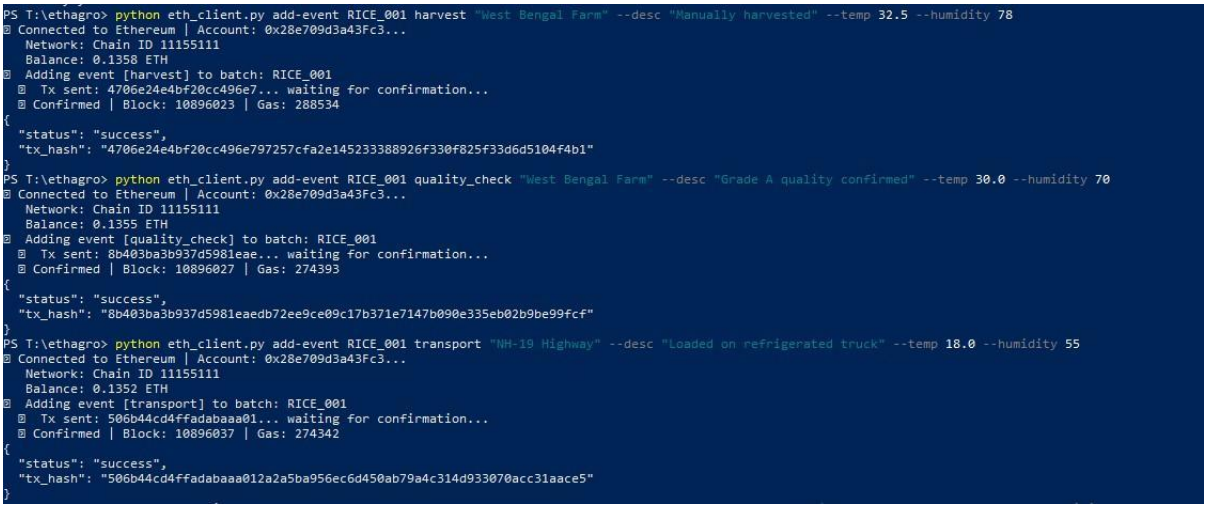

Figure 3.harvest, quality_check and transport events for RICE_001 (Blocks 10,896,023–10,896,037).

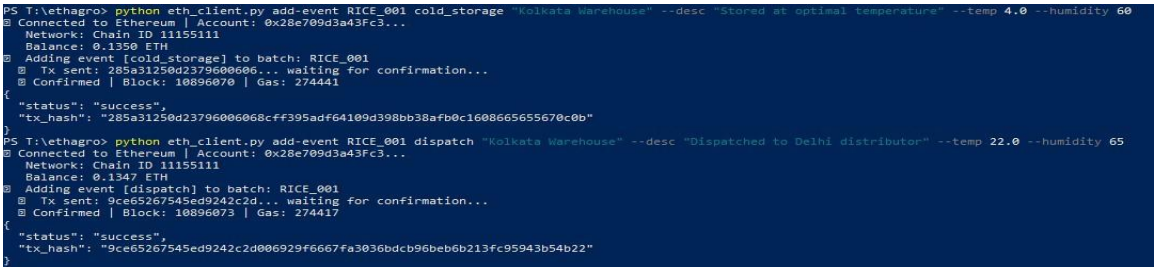

Figure 4. cold_storage and dispatch events for RICE_001 (Blocks 10,896,070–10,896,073).

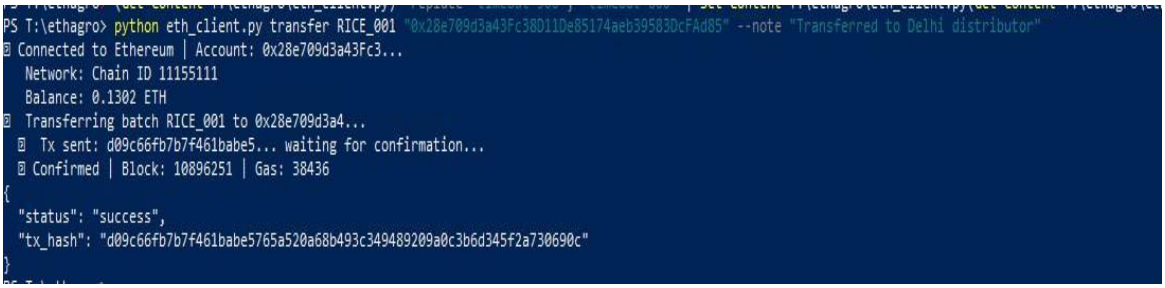

Figure 5.Ownership transfer of RICE_001, confirmed Block 10,896,251.

```
PS T:\ethagro> python eth_client.py split RICE_001 RICE_001_DELHI RICE_001_MUMBAI 600 400 --reason "Split for two distributors"
✓ Connected to Ethereum | Account: 0x28e709d3a43Fc3...
   Network: Chain ID 11155111
   Balance: 0.1297 ETH
✂  Splitting RICE_001 → RICE_001_DELHI(600) + RICE_001_MUMBAI(400)
  ✓  Tx sent: 28c42ce957414720b990... waiting for confirmation...
  ✓ Confirmed | Block: 10896257 | Gas: 691659
```

Figure 6.splitBatch: RICE_001 (1,000 kg) → RICE_001_DELHI (600 kg) + RICE_001_MUMBAI (400 kg), confirmed Block 10,896,257

```
PS T:\ethagro> python eth_client.py oracle RICE_001_DELHI temperature 4.2 C 95 SENSOR_001
✓ Connected to Ethereum | Account: 0x28e709d3a43Fc3...
   Network: Chain ID 11155111
   Balance: 0.1191 ETH
✓  Oracle: [temperature=4.2C, conf=95%] → RICE_001_DELHI
  ✓  Tx sent: 602a236b9c8b9d555544... waiting for confirmation...
  ✓ Confirmed | Block: 10896279 | Gas: 388335
{
  "status": "success",
  "tx_hash": "602a236b9c8b9d5555441213dcad09061d419aaa279df54d99cf6922e1f53df0"
}
PS T:\ethagro> python eth_client.py oracle RICE_001_DELHI humidity 62.0 percent 90 SENSOR_002
✓ Connected to Ethereum | Account: 0x28e709d3a43Fc3...
   Network: Chain ID 11155111
   Balance: 0.1108 ETH
✓  Oracle: [humidity=62.0percent, conf=90%] → RICE_001_DELHI
  ✓  Tx sent: 99f822738ce87e11e82d... waiting for confirmation...
  ✓ Confirmed | Block: 10896281 | Gas: 323429
{
  "status": "success",
  "tx_hash": "99f822738ce87e11e82d3f78a840c143e360a124a391ee5d3f2e0b361ebacfa5"
}
PS T:\ethagro> python eth_client.py oracle RICE_001_MUMBAI temperature 3.8 C 92 SENSOR_003
✓ Connected to Ethereum | Account: 0x28e709d3a43Fc3...
   Network: Chain ID 11155111
   Balance: 0.1040 ETH
✓  Oracle: [temperature=3.8C, conf=92%] → RICE_001_MUMBAI
  ✓  Tx sent: b5a66a1db7a8a2e6d6dd... waiting for confirmation...
  ✓ Confirmed | Block: 10896282 | Gas: 354147
{
  "status": "success",
  "tx_hash": "b5a66a1db7a8a2e6d6ddbb19d38ba3fed3f3b73820a0c46bda3ba28ad261ad95"
}
```

Figure 7.Three oracle submissions - temperature and humidity for RICE_001_DELHI, temperature for RICE_001_MUMBAI, all confidence ≥ 90 and marked verified.

```
PS T:\ethagro> python eth_client.py init-batch CORN_001 "Organic Corn" "Punjab Farm" 300 kg
✓ Connected to Ethereum | Account: 0x28e709d3a43Fc3...
   Network: Chain ID 11155111
   Balance: 0.0966 ETH
✓  Initializing batch: CORN_001
  ✓  Tx sent: 957733210e37f9e80164... waiting for confirmation...
  ✓ Confirmed | Block: 10896287 | Gas: 286779
{
  "status": "success",
  "tx_hash": "957733210e37f9e801640327ea58253c06e09c6bdd951603a5f44e336cd54732",
  "block": 10896287
}
PS T:\ethagro> python eth_client.py init-batch CORN_002 "Organic Corn" "Punjab Farm" 200 kg
✓ Connected to Ethereum | Account: 0x28e709d3a43Fc3...
   Network: Chain ID 11155111
   Balance: 0.0907 ETH
✓  Initializing batch: CORN_002
  ✓  Tx sent: a66ad512506419fc475c... waiting for confirmation...
  ✓ Confirmed | Block: 10896290 | Gas: 286767
{
  "status": "success",
  "tx_hash": "a66ad512506419fc475c2b08779ad2ee9989be4eb19ae5c9a07ed37ca51587d0",
  "block": 10896290
}
PS T:\ethagro> python eth_client.py merge CORN_001 CORN_002 CORN_MERGED --reason "Consolidated for export"
✓ Connected to Ethereum | Account: 0x28e709d3a43Fc3...
   Network: Chain ID 11155111
   Balance: 0.0857 ETH
✓  Merging CORN_001 + CORN_002 → CORN_MERGED (total: 500)
  ✓  Tx sent: 8f0129d96285020b6821... waiting for confirmation...
  ✓ Confirmed | Block: 10896291 | Gas: 369154
{
  "status": "success",
  "tx_hash": "8f0129d96285020b68215be3100b0f28eb5dc790a55915dde2d78fb10d851ade"
}
```

Figure 8. init-batch for CORN_001 and CORN_002 followed by mergeBatches → CORN_MERGED, confirmed Block 10,896,291.

5.2 Cumulative Deployment State

Beyond the primary experiment, Table 2 reports the system's cumulative statistics as recorded by its own indexer and export function, cross-checked directly against the SQLite database supplied with this study. These figures span both deployment phases described in Section 4.2 and therefore include five product batches (Wheat, Kiwi, Mango

×2, Grapes, Apple) beyond the Rice and Corn batches detailed in Table 1. The system dashboard at an intermediate point in the deployment is shown in Fig. 9.

Table 2.Cumulative on-chain deployment statistics, cross-checked against agrichain_eth.db. The zero split count is investigated in Section 5.4 - it reflects an off-chain indexing gap, not the absence of on-chain splits (cf. Fig. 6, Table 1).

Metric (as recorded by the local indexer)	Value
Root-level batches indexed	9
Distinct commodities represented	7 (Wheat, Kiwi, Mango, Grapes, Apple, Rice, Corn)
Total events logged	16
Total merges recorded	3 (Wheat pair, Mango pair, Corn pair)
Total splits recorded	0 - see Section 5.4
Total oracle submissions	8
Oracle submissions verified (confidence ≥ 80)	8 / 8 (100%)
Total ownership transfers	3

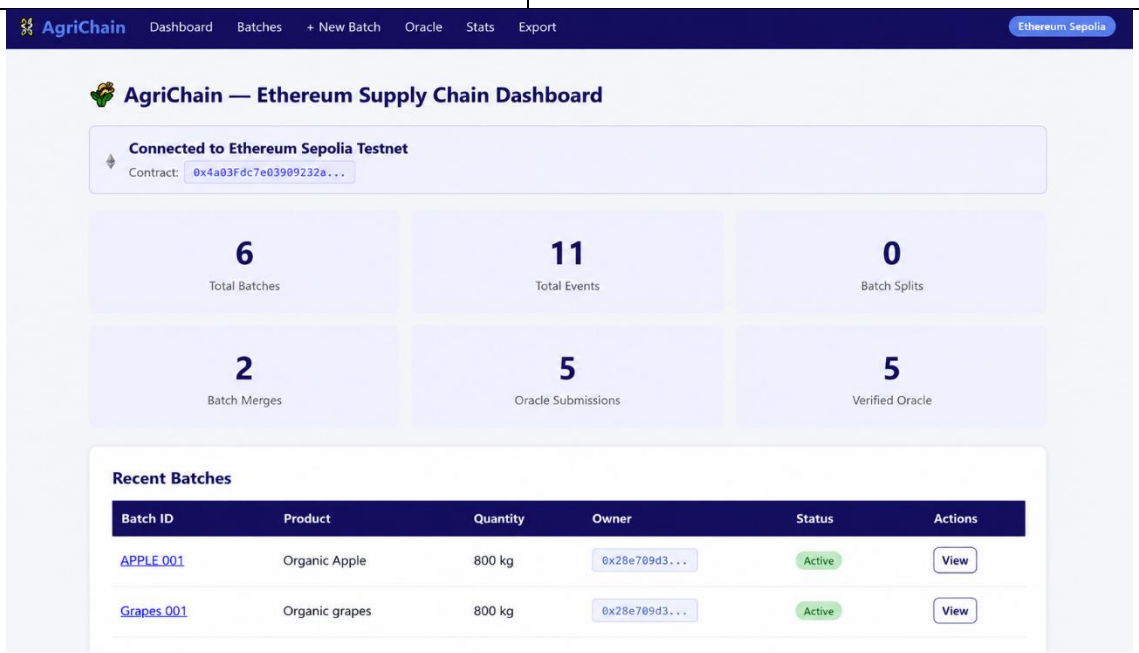

Figure 9. AgriChain dashboard, captured at an earlier point in the deployment (6 batches / 11 events / 2 merges / 5 oracle submissions) before the RICE_001 and CORN_001/CORN_002 operations of Table 1 were executed. Shown for illustration of the presentation layer; later figures for the exact current totals are given in Table 2.

5.3 Gas Cost Analysis

Fig. 10 and Table 3 summarise gas consumption per contract function, computed directly from the measured values in Table 1 (mean, minimum and maximum across all confirmed calls to that function). All transactions were confirmed at zero monetary cost, since Sepolia ETH is obtained free of charge from a testnet faucet; the illustrative USD figures in Table 3 convert these real gas measurements at Ethereum Mainnet conditions prevailing at the time of writing an average gas price of 0.086 gwei and an ETH price of USD 1,576.93, both from Etherscan on 5 July 2026

[21] and are provided only to give an order-of-magnitude sense of production cost; both quantities fluctuate continuously with network demand and market conditions and Ethereum gas prices in particular have historically ranged from a small fraction of a gwei to well over 100 gwei during periods of high network congestion, so these figures should not be read as a durable cost forecast.

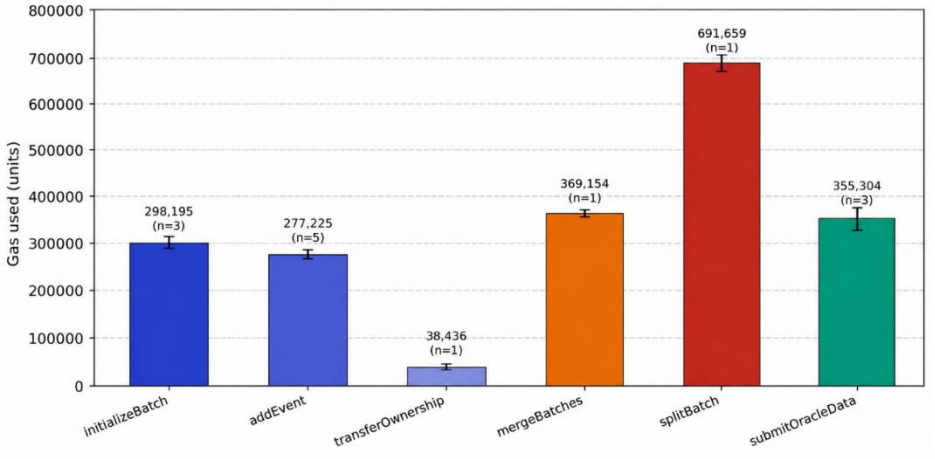

Figure 10. Measured gas consumption per contract function (confirmed Sepolia transactions; error bars show min-max range)

Table 3.Gas cost per function (mean/min/max computed from Table 1) with illustrative Mainnet-equivalent cost at 0.086 gwei and ETH = USD 1,576.93 (Etherscan, 5 July 2026 [21]). splitBatch is the most expensive single call, consistent with it writing two new Batch structs in addition to updating the parent.

Function	Mean gas	Min–max gas	Illustrative cost (ETH)	Illustrative cost (USD)
initializeBatch	298,195	286,767 - 321,039	0.00002564	$0.040
addEvent	277,225	274,342 - 288,534	0.00002384	$0.038
transferOwnership	38,436	38,436	0.00000331	$0.005
mergeBatches	369,154	369,154	0.00003175	$0.050
splitBatch	691,659	691,659	0.00005948	$0.094
submitOracleData	355,304	323,429 - 388,335	0.00003056	$0.048
Full 14-tx primary experiment	4,445,872 (sum)	-	0.00038235	$0.603

5.4 An On-Chain / Off-Chain Synchronisation Defect

Cross-referencing Table 1 against Table 2 reveals an apparent contradiction. Fig. 6 shows the splitBatch transaction for RICE_001 confirmed at Block 10,896,257 (Gas 691,659) and the three subsequent oracle submissions in Fig. 7 are addressed to RICE_001_DELHI and RICE_001_MUMBAI - batch identifiers that can only be valid targets for submitOracleData if they already exist on-chain, since that function is gated by the onlyExists modifier (Section 3.3). The split therefore unambiguously occurred on-chain. Yet Table 2 reports zero recorded splits and direct inspection of agrichain_eth.db confirms that RICE_001 still carries status = 'Active' with current_quantity = 1,000 in the local database and that no row exists for RICE_001_DELHI or RICE_001_MUMBAI in the batches table. The same pattern recurs earlier in the deployment history: the oracle_data table contains verified readings against WHEAT_001A, KIWI_001_MUMBAI and KIWI_001_PUNE again, identifiers that must exist on-chain to have

accepted an oracle submission none of which appear as rows in the local batches table and the splits table is empty across the entire deployment history, not only for RICE_001.

We traced this to a concurrency defect in eth_client.py's EthIndexer.index_split() method. That method opens a SQLite connection and, within its own open write transaction, first inserts a row into the splits table and updates the parent batch's status and then calls self.index_batch() twice, once for each child batch, to record the two newly created batches. Critically, index_batch() independently opens its own new SQLite connection via the same self._conn() helper, rather than reusing the connection already open in index_split(). Under SQLite's default rollback-journal mode, a second connection attempting to write to the same database file while a first connection holds an uncommitted write transaction cannot proceed; it blocks until the configured busy-timeout elapses and then raises a 'database is locked' error. Because this exception is raised inside the first connection's still-open Python with block, it propagates outward and causes that block's implicit rollback undoing the splits-table insert and the parent-status update that had already executed, in addition to preventing the child-batch inserts from ever committing. The net effect is that a fully successful on-chain split leaves no trace whatsoever in the local index: not the split record, not the parent's updated status and not the two new child batches.

We did not treat this as a plausible hypothesis to merely assert; we reproduced it directly. Using the same connection pattern as EthIndexer an outer connection performing a write, with a nested inner connection to the same file attempting a second write before the outer transaction commits an isolated test against a throwaway SQLite database raised exactly the predicted sqlite3.OperationalError ('database is locked') and inspection of both tables afterward showed them empty, confirming that the outer, apparently-already-executed insert had been rolled back by the failure of the nested call. This matches the observed production data precisely: an on-chain event that unquestionably occurred, entirely absent from every downstream artefact - the local database, the dashboard (Fig. 9) and the JSON export - that is derived from that database rather than from the chain itself.

The corresponding merge path, EthIndexer.index_merge(), does not exhibit the crash, because it does not call index_batch() from within its own open connection - it only inserts into the merges table and updates the two existing source rows, both within a single connection. This is why Table 2 correctly reports 3 merges. However, index_merge() also never inserts a row for the newly created merged batch itself (e.g., CORN_MERGED), so while the merge event and the status change of the two source batches are correctly indexed, the resulting merged batch is, like the split children, invisible to any query that reads only the local database rather than the chain a related but distinct omission with the same practical consequence.

The broader, generalisable lesson is that for any blockchain application pairing an authoritative on-chain ledger with a local off-chain index for fast querying an architecture used by the great majority of production blockchain applications, including this one on-chain correctness and off-chain completeness are separable properties and neither implies the other. A batch, a split, or a merge can be entirely valid and permanently recorded on a public ledger while being completely absent from every dashboard, report and statistic derived from a local mirror of that ledger, if the mirror's write logic does not treat each logical on-chain operation as a single atomic local transaction. A stakeholder who queried this system only through its dashboard or JSON export rather than by directly inspecting the chain, as this evaluation did would have concluded, incorrectly, that RICE_001 was never split and that RICE_001_DELHI and RICE_001_MUMBAI do not exist, when in fact both are valid, oracle-bearing batches on Sepolia at the time of writing.

6. Discussion

6.1 Remediating the Synchronisation Defect

The defect identified in Section 5.4 has a straightforward fix: index_split() (and for the merged-batch omission, index_merge()) should perform every write belonging to one logical on-chain operation within a single open connection and transaction, passing that connection or an equivalent cursor into any helper it calls, rather than allowing a helper to open a second, independent connection to the same file. Enabling SQLite's write-ahead-logging (WAL) journal mode would also relax the single-writer restriction that triggers the lock, though restructuring the call chain to use one connection per logical operation is the more robust fix, since it removes the race condition rather than narrowing the window in which it can occur. We note this not to single out a single line of code, but because the pattern a convenience helper method that transparently opens its own database connection is easy to introduce in any indexer of this kind and the resulting failure mode (a silent, exception-triggered rollback with no user-visible error in the CLI output - the terminal showed only a successful confirmation (Fig. 6), giving no indication that the local index

had silently rolled back) is exactly the kind of defect that is difficult to notice without the cross-referencing exercise performed in Section 5.4.

6.2 Cost and Scalability Considerations

The measured gas costs in Table 3 are modest in absolute terms the entire 14-transaction primary experiment corresponds to an illustrative cost of under one US dollar at current Mainnet gas prices - but gas prices are volatile and splitBatch's 691,659 gas is nearly eighteen times an ownership transfer's 38,436 gas, since it writes two complete new Batch structs to contract storage in addition to updating the parent. For a deployment tracking many thousands of batch operations, cumulative gas cost, not any single transaction, would be the operative planning concern and Ethereum Layer-2 networks or a permissioned deployment would be natural avenues for reducing it; we do not report a quantitative comparison here, since doing so would require an equivalent, equally rigorously measured deployment, which was outside the scope of the evidence available for this paper.

6.3 Security and Access-Control Observations

Three design characteristics of the current contract are worth stating plainly rather than leaving implicit. First, submitOracleData is gated only by onlyExists, with no ownership or allow-list restriction (Section 3.3): any Ethereum address can submit a sensor-style reading against any existing batch and the recorded validator address establishes non-repudiation of who submitted a value but does not, by itself, establish that the submitter was an authorised sensor operator for that batch. A production deployment would likely want a registrable allow-list of validator addresses per batch or per operator organisation, consistent with the oracle-authentication concerns raised more generally by Caldarelli [18]. Second, the 32-byte identifier-length check applied in initializeBatch is not applied to the child identifiers created by splitBatch or the merged identifier created by mergeBatches, so a split or merge could, in principle, create a batch identifier that would have been rejected had it been supplied directly to initializeBatch; we observed no case of this occurring in the evidence set, but the inconsistency is worth closing for defence in depth. Third, as noted in Section 3.2, the BatchStatus.Recalled state is defined but never assigned by any function in the current contract; it appears to be a placeholder for a future recall or quarantine workflow. None of these three observations were triggered in the transactions evaluated in this paper and we did not observe or capture an on-chain revert transaction demonstrating the mass-balance check under an actual violation attempt; the check's presence and logic are verified by direct source-code inspection (Section 3.3), but an empirical revert-transaction receipt, of the kind captured for every successful operation in Table 1, was not part of the available evidence and so is not claimed here.

6.4 Limitations

This evaluation has four main limitations. First, it reflects a single-operator demonstration – every transaction in Tables 1 and 2 originates from one wallet address rather than a multi-party deployment with independent farmers, distributors and retailers each controlling their own keys, which is the target production scenario. Second, no concurrent-load or repeated-trial throughput testing was performed or is reported; the gas and timing figures in this paper describe individual, sequentially executed transactions rather than system behaviour under contention. Third, the evaluation is limited to a public test network; Sepolia mirrors Mainnet's execution semantics and consensus family but not its real economic finality or fee-market conditions under load. Fourth, the source comments in eth_client.py indicate that an equivalent Solana-based implementation of the same workflow exists as part of the authors' broader research programme; a rigorous, like-for-like comparison between the two platforms would be genuinely valuable, but doing so credibly requires an equally carefully measured Solana deployment, which was not part of the evidence available for this paper and we therefore do not report comparative figures for it here.

6.5 Future Work

Immediate next steps are: (i) applying the connection-scoping fix described in Section 6.1 and re-verifying that split and merge operations are then fully reflected in the local index and JSON export; (ii) introducing role-based access control (an allow-list of validator addresses) for submitOracleData; (iii) applying the same identifier-length validation across initializeBatch, splitBatch and mergeBatches; (iv) wiring the Recalled state into an explicit recall/quarantine function with its own access control and event; (v) conducting a controlled, repeated-trial throughput

and latency benchmark under concurrent load, ideally across several Layer-2 networks in addition to Ethereum L1; and (vi), once a comparably rigorous Solana deployment is available, a direct, same-workload comparison between the two platforms using the measurement methodology demonstrated in Section 5 of this paper.

7. Conclusion

This paper presented AgriChain, a six-function Ethereum smart-contract system for agricultural batch traceability and evaluated it using only independently verifiable evidence: real Sepolia block numbers, gas measurements and transaction hashes for a fully traced 14-transaction primary experiment and the system's cumulative two-phase deployment history, an independently recompiled bytecode size confirming EIP-170 compliance and a genuine defect uncovered by cross-referencing the on-chain evidence against the system's own database rather than trusting either source in isolation. That defect a nested-SQLite-connection concurrency fault that silently erases confirmed on-chain split operations from the off-chain index was root-caused to a specific code pattern and independently reproduced and we believe its underlying lesson, that on-chain correctness and off-chain completeness are separable properties requiring separate engineering attention, generalises well beyond this one deployment. Combined with a candid account of the system's current access-control and validation gaps, this paper aims to demonstrate that a rigorously and honestly reported blockchain traceability prototype including what does not yet work correctly is more useful to the research community and to practitioners than a polished account that omits it.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Food and Agriculture Organization of the United Nations (FAO). Global Food Losses and Food Waste — Extent, Causes and Prevention. FAO, Rome, 2011. [Online]. Available: https://www.fao.org/3/mb060e/mb060e.pdf.
2. Feng, H.; Wang, X.; Duan, Y.; Zhang, J.; Zhang, X. Applying blockchain technology to improve agri-food traceability: A review of development methods, benefits and challenges. J. Clean. Prod. 2020, 260, 121031. https://doi.org/10.1016/j.jcleprod.2020.121031.
3. Morabito, V. Business Innovation Through Blockchain: The B³ Perspective. Springer, Cham, 2017. https://doi.org/10.1007/978-3-319-48478-5.
4. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf.
5. Wood, G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Ethereum Project Yellow Paper, 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf.
6. Buterin, V. A Next-Generation Smart Contract and Decentralized Application Platform. Ethereum White Paper, 2013. [Online]. Available: https://ethereum.org/en/whitepaper/.
7. Tian, F. An agri-food supply chain traceability system for China based on RFID and blockchain technology. In Proceedings of the 13th International Conference on Service Systems and Service Management (ICSSSM), Kunming, China, 24–26 June 2016; pp. 1–6. https://doi.org/10.1109/ICSSSM.2016.7538424.
8. Tian, F. A supply chain traceability system for food safety based on HACCP, blockchain and Internet of Things. In Proceedings of the International Conference on Service Systems and Service Management (ICSSSM), Dalian, China, 16–18 June 2017; pp. 1–6. https://doi.org/10.1109/ICSSSM.2017.7996119.
9. Wang, Y.; Han, J.H.; Beynon-Davies, P. Understanding blockchain technology for future supply chains: A systematic literature review and research agenda. Supply Chain Manag. Int. J. 2019, 24, 62–84. https://doi.org/10.1108/SCM-03-2018-0148.
10. Kouhizadeh, M.; Saberi, S.; Sarkis, J. Blockchain technology and the sustainable supply chain: Theoretically exploring adoption barriers. Int. J. Prod. Econ. 2021, 231, 107831. https://doi.org/10.1016/j.ijpe.2020.107831.
11. Ethereum Improvement Proposal 170 (EIP-170): Contract Code Size Limit, 2016. [Online]. Available: https://eips.ethereum.org/EIPS/eip-170.
12. Olsen, P.; Borit, M. How to define traceability. Trends Food Sci. Technol. 2013, 29, 142–150. https://doi.org/10.1016/j.tifs.2012.10.003.
13. Kshetri, N. Blockchain's roles in meeting key supply chain management objectives. Int. J. Inf. Manag. 2018, 39, 80–89. https://doi.org/10.1016/j.ijinfomgt.2017.12.005.
14. Kamble, S.S.; Gunasekaran, A.; Sharma, R. Modeling the blockchain-enabled traceability in agriculture supply chain. Int. J. Inf. Manag. 2020, 52, 101967. https://doi.org/10.1016/j.ijinfomgt.2019.05.023.
15. Sunny, J.; Undralla, N.; Pillai, V.M. Supply chain transparency through blockchain-based traceability: An overview with demonstration. Comput. Ind. Eng. 2020, 150, 106895. https://doi.org/10.1016/j.cie.2020.106895.

16. Sharma, A.; Bhatia, T.; Sharma, A.; Aggarwal, P. Secure and Traceable Decentralized Agri-Food Supply Chain Framework Using Ethereum Blockchain and IPFS Platform. Trans. Emerg. Telecommun. Technol. 2025, 36, e70188. https://doi.org/10.1002/ett.70188.
17. Ethereum Foundation. The Merge. 2022. [Online]. Available: https://ethereum.org/en/upgrades/merge/.
18. Caldarelli, G. Understanding the Blockchain Oracle Problem: A Call for Action. Information 2020, 11, 509. https://doi.org/10.3390/info11110509.
19. Reyna, A.; Martín, C.; Chen, J.; Soler, E.; Díaz, M. On blockchain and its integration with IoT: Challenges and opportunities. Future Gener. Comput. Syst. 2018, 88, 173–190. https://doi.org/10.1016/j.future.2018.05.046.
20. Ellis, S.; Juels, A.; Nazarov, S. ChainLink: A Decentralized Oracle Network. Chainlink White Paper, 2017. [Online]. Available: https://research.chain.link/whitepaper-v1.pdf.
21. Etherscan. Ethereum gas tracker and ETH price. [Online]. Available: https://etherscan.io/gastracker [accessed 5 July 2026].