

## Article

# A Kubernetes-Deployed Tamper-Evident Media-Evidence Provenance Pipeline with Hybrid Blockchain/IPFS Anchoring and Queue-Mediated Ingress

Haoliang Wang <sup>1,2,\*</sup>, Zarina Shukur <sup>1</sup>, Khairul Akram Zainol Ariffin <sup>1</sup>  and Lili Wang <sup>2,\*</sup> 

<sup>1</sup> Center for Cyber Security, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Bangi 43600, Selangor, Malaysia; zarinashukur@ukm.edu.my (Z.S.); k.akram@ukm.edu.my (K.A.Z.A.)

<sup>2</sup> School of Artificial Intelligence, Dongguan City University, Dongguan 523419, China

\* Correspondence: p109025@siswa.ukm.edu.my (H.W.); wangll@dgcu.edu.cn (L.W.)

## Abstract

High-stakes online assessment produces suspicious-event records, yet storage placement and burst admission remain insufficiently characterized. This article presents a Kubernetes-deployed provenance pipeline integrating Hyperledger Fabric, IPFS, SHA-256 verification, and RabbitMQ ingress. Media objects are retained in IPFS, while compact semantics, CIDs, and verification anchors are committed to Fabric. At 5 TPS, five 300-transaction runs completed under both direct large-payload and compact-anchor ledger conditions. At 20 TPS, compact anchoring of a pre-retained IPFS object sustained  $19.40 \pm 0.00$  TPS with  $0.22 \pm 0.00$  s mean ledger latency; direct large-payload submission achieved  $14.20 \pm 0.40$  TPS with  $31.62 \pm 3.35$  s latency. A write-load sweep identified 100 TPS as the transition point; higher loads were delay-dominated. Three 1000-VU PTS runs reduced mean response time from  $4.76 \pm 0.78$  s to  $2.40 \pm 0.02$  s, with similar failure rates. In three 30-message consumer-enabled runs, all 90 messages reached IPFS retention, Fabric commit, query visibility, and manual acknowledgement without retry, producer failure, or dead-letter outcome. Mean queue-drain and total completion times were  $61.80 \pm 0.71$  s and  $62.99 \pm 0.15$  s. The results isolate Fabric transaction-path payload cost and distinguish queue acceptance from downstream completion.

**Keywords:** tamper-evident provenance; media evidence; Hyperledger Fabric; IPFS; Kubernetes; RabbitMQ; queue-mediated ingress; online examination; auditability



Academic Editor:

Aryya Gangopadhyay

Received: 28 June 2026

Revised: 29 July 2026

Accepted: 5 August 2026

Published: 6 August 2026

**Copyright:** © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

## 1. Introduction

High-stakes online assessment increasingly relies on upstream proctoring and anomaly-detection modules to generate suspicious-event evidence for later review. Such evidence may include screen-capture images, detector outputs, timestamps, examination context, and storage references. Research on online examinations has examined cheating patterns and countermeasures [1], automated cheating-detection and image-hashing methods that produce candidate suspicious-event records [2,3], and webcam-based or online-proctoring practices that support subsequent integrity review [4,5]. Blockchain-supported educational systems have addressed the integrity of credentials and examination records [6–8]. Media-bearing suspicious-event records nevertheless present a distinct workload because they combine structured event semantics, large evidence objects, privacy-sensitive content, and institutional review requirements.

This distinction creates a systems problem with direct evidential consequences. Suspicious-event evidence may be used in post-exam review, appeal handling, disciplinary decisions, and institutional quality assurance. A provenance service must therefore preserve the linkage among the media object, examination context, detector output, verification value, and submission time. Loss of the retained object, alteration of its context, ambiguous verification results, or delayed admission during a concentrated burst can weaken the record as reviewable process evidence.

The design challenge arises from the coupling of integrity, storage placement, and service admission. Content-addressed storage permits evidence objects to remain outside the ledger path [9], whereas Fabric studies show that payload size influences endorsement, ordering, validation, and commit cost [10,11]. Work on decentralized storage, evidence preservation, judicial evidence, and document-version provenance supports off-chain retention through verifiable references [12–15]. At the service boundary, synchronous submission exposes provenance-processing delay to clients during unified examination starts, concentrated submission windows, and post-exam evidence uploads. Cloud-native service coordination, queue-based workflows, and edge-workload orchestration motivate a separate message-publication boundary for burst traffic [16–18]. Object-storage performance and autoscaling studies further inform the downstream retention and service-capacity boundary [19,20]. These mechanisms have largely been evaluated in isolation; their interaction across storage, ledger, and ingress boundaries remains insufficiently characterized for media-rich educational process evidence.

This article develops a Kubernetes-deployed tamper-evident media-evidence provenance pipeline that anchors compact metadata and SHA-256 evidence values on Hyperledger Fabric, retains evidence objects off-chain through IPFS, and evaluates direct synchronous submission, producer-only RabbitMQ publication, and consumer-enabled completion under fixed deployment resources. Online examination integrity provides the empirical setting because suspicious events combine structured context, privacy-sensitive media, subsequent review, and burst-prone submission. The evaluated pipeline begins after event generation; the upstream detector supplies event records under a fixed configuration, and the experiments focus on provenance storage, evidence verification, ledger behavior, and ingress coordination.

The evaluation is organized around three research questions derived from the interaction among media payload placement, permissioned-ledger commit behavior, and service-boundary burst admission. RQ1 asks how direct large-payload and compact-anchor Fabric transaction paths differ in throughput, latency, and successful completion under the same offered load. RQ2 asks where the fixed-topology provenance write path changes from an operational region to a delay-dominated saturation region as offered load increases. RQ3 asks how persistent-message publication to RabbitMQ changes API-level request processing, mean response time, residual failure rate, and tail latency during a synchronized burst, and how admitted messages progress through IPFS retention, Fabric commit, query visibility, manual acknowledgement, and queue drain under one controlled consumer.

This article makes four contributions. First, it conceptualizes online-examination media evidence as a process-evidence workload in which object size, semantic context, verification boundary, and burst arrival must be evaluated jointly. Second, it implements a three-plane pipeline that separates evidence generation, Fabric/IPFS provenance anchoring, and queue-mediated ingress while preserving an auditable binding among event semantics, SHA-256 evidence values, content identifiers, and transaction context. Third, it implements two verification paths over the same SHA-256 anchor: candidate-object verification for a supplied object and CID-level byte audit for the exact object retrieved from IPFS. Fourth, it provides repeated measurements of ledger payload placement, provenance-write sat-

uration, HTTP-to-queue publication, and consumer-enabled completion, with separate metrics for ledger completion, API outcomes, queue drain, stage latency, retries, failures, and total completion.

The remainder of the article is structured as follows. Section 2 reviews media-evidence workloads, tamper-evident provenance, and queue-mediated service coordination. Section 3 describes the system design, software architecture, threat model, and evaluation protocol. Section 4 presents functional validation and performance results. Section 5 interprets the findings and discusses engineering implications, external validity, security and privacy boundaries, and future research. Section 6 concludes the article.

## 2. Related Work

### 2.1. Online Assessment Evidence Workloads

Online assessment research has characterized cheating and proctoring evidence as a basis for later review [1]. Automated cheating-detection and image-hashing methods produce machine-generated candidate events for institutional audit [2,3], while webcam-based and online-proctoring studies show how captured observations support reviewable examination-integrity records [4,5]. These studies establish the upstream evidence context; the resulting media records must still be preserved, linked to their event semantics, and admitted under burst-prone service conditions.

From a cloud-system perspective, event generation is therefore only the first stage. A useful evidence record must preserve contextual semantics, retain a retrievable object reference, support later integrity checking, and remain serviceable during burst arrivals. These requirements connect educational evidence generation with cloud storage, service admission, and tamper-evident provenance.

### 2.2. Tamper-Evident Educational Records and Process Evidence

Blockchain applications in education have addressed academic-integrity records and online-exam result reliability [6,7], while broader educational blockchain research has examined verifiable credentials, learning records, and institutional trust [8]. These studies establish tamper-evident records, accountable institutional participation, and verifiable academic artifacts as relevant design goals.

Media-rich process evidence has a different temporal and structural profile. Credentials are compact outcome artifacts that are repeatedly verified after issuance. Suspicious-event evidence is formed during a live service process, combines structured context with unstructured media, and is typically consulted when a later review requires reconstruction of an event. This distinction motivates a storage and verification design that preserves event semantics while keeping media objects outside the ledger path.

### 2.3. Hybrid Anchoring and Object-Retention Provenance

Content-addressed storage provides the object-reference basis for retaining media outside the ledger path [9]. Permissioned-blockchain and Fabric performance studies motivate controlled institutional anchoring and careful treatment of transaction-path cost [10,11]. Decentralized storage for cloud-native edge settings [12], IPFS-based evidence preservation [13], judicial evidence preservation [14], and document-version provenance [15] demonstrate how compact references and integrity values can be combined with off-chain object retention.

The evaluated pipeline applies this pattern to media-rich educational evidence. The chain record binds the assessment context, detector output, candidate-verification value, and object reference into a reviewable provenance entry. The off-chain layer carries the object size, while the ledger preserves the semantic and verification anchor. This separation

reduces ledger payload burden but retains the need for explicit object-retention and byte-audit procedures.

#### 2.4. Kubernetes Service Coordination and Queue-Mediated Ingress

Distributed service architectures separate heterogeneous responsibilities across cloud-native service boundaries [16]. Queue-based workflow coordination can absorb bursts, expose backlog, and decouple client-facing acceptance from slower downstream work [17]. Workload orchestration [18], object-storage performance [19], and autoscaling research [20] further show that service capacity and telemetry shape the boundary between request admission and completed processing. These principles are relevant when evidence arrives faster than a provenance substrate can retain objects and commit anchors.

RabbitMQ provides the explicit message-publication boundary in the evaluated pipeline. The asynchronous gateway publishes each evidence task as a persistent message to a durable queue and returns HTTP 202 after the RabbitMQ publish call completes without error. The producer-side ingress measurements characterize HTTP-to-queue publication, while separate consumer-enabled completion runs follow admitted tasks through IPFS retention, Fabric commit, query visibility, and manual acknowledgement.

#### 2.5. Positioning of the Evaluated Pipeline

Prior work provides the individual mechanisms for event generation, educational-record integrity, blockchain/object-storage provenance, and queue-based service coordination. The contribution of this study lies in their composition and evaluation for media-rich process evidence: the ledger path receives either compact semantics or a large media field, while the ingress evaluation separates producer-side HTTP-to-queue publication from consumer-enabled downstream completion. Two controlled architectural contrasts, one saturation analysis and one completion experiment, preserve separate metric semantics for ledger completion, API admission, queue drain, and completed evidence processing. Table 1 positions these research lines relative to the evaluated pipeline.

**Table 1.** Representative related work and positioning of the evaluated pipeline.

| Study/Line of Work                                                                                                                                | Technical Emphasis                                                                        | Evidence-Pipeline Implication                                       | Role in This Article                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|---------------------------------------------------------------------|-----------------------------------------------|
| Noorbehbahani et al. [1]; Hylton et al. [4]; Dendir and Maxwell [5]                                                                               | Online proctoring, cheating detection, and reviewable assessment evidence                 | Online examinations can yield reviewable suspicious-event evidence  | Motivates process-evidence governance         |
| Kaddoura and Gumaï [2]; Yaqub et al. [3]                                                                                                          | Automated cheating detection and image-hashing anomaly detection                          | Automated methods can produce candidate evidence events at scale    | Provides upstream event-generation context    |
| Cardenas-Quispe and Pacheco [6]; Abdelsalam et al. [7]; Alammary et al. [8]                                                                       | Academic integrity, online-exam result reliability, and educational blockchain            | Tamper-evident records can support institutional trust              | Extends record governance to process evidence |
| Trautwein et al. [9]; Androulaki et al. [10]; Thakkar et al. [11]; Zang et al. [12]; Rani et al. [13]; Liu and Zheng [14]; Nizamuddin et al. [15] | IPFS, Fabric, and off-chain evidence retention                                            | Media can remain off-chain while compact references stay auditable  | Supports hybrid anchoring                     |
| Deng et al. [16]; Cao et al. [17]; Vaño et al. [18]; Durner et al. [19]; Alharthi et al. [20]                                                     | Cloud-native services, queues, orchestration, object-storage performance, and autoscaling | Burst admission and durable completion require separate measurement | Supports ingress evaluation                   |
| Deng et al. [21]; Wu et al. [22]                                                                                                                  | Verifiable provenance queries and learned blockchain indexes                              | Repository-scale evidence records require efficient retrieval       | Positions query scaling as future work        |

Provenance verification and query efficiency form a subsequent stage of the evidence lifecycle. vProChain provides efficient provenance verification and query support for Industrial Internet of Things provenance, while TELEX accelerates rich blockchain queries through a two-level learned index in an enclave-based setting [21,22]. These studies address verification and retrieval after provenance records have accumulated. The present pipeline examines the earlier operational stage in which media-bearing evidence is admitted, retained, and anchored before entering a large queryable corpus.

Identity binding and repository-scale query form complementary design layers. Decentralized identity can strengthen role and authority verification, while learned or verifiable indexing can improve large-scale provenance retrieval. The evaluated pipeline concentrates on the interaction among storage placement, ledger behavior, and ingress admission for media-bearing evidence.

### 3. System Design and Evaluation Protocol

#### 3.1. Problem Formulation and Scope

The system targets institutional online examinations in which candidates operate in uncontrolled physical spaces and detection, provenance, and coordination services run in an institution-controlled cloud environment. The scope focuses on suspicious-event evidence after detection: preservation, verification, and admission into the provenance pipeline.

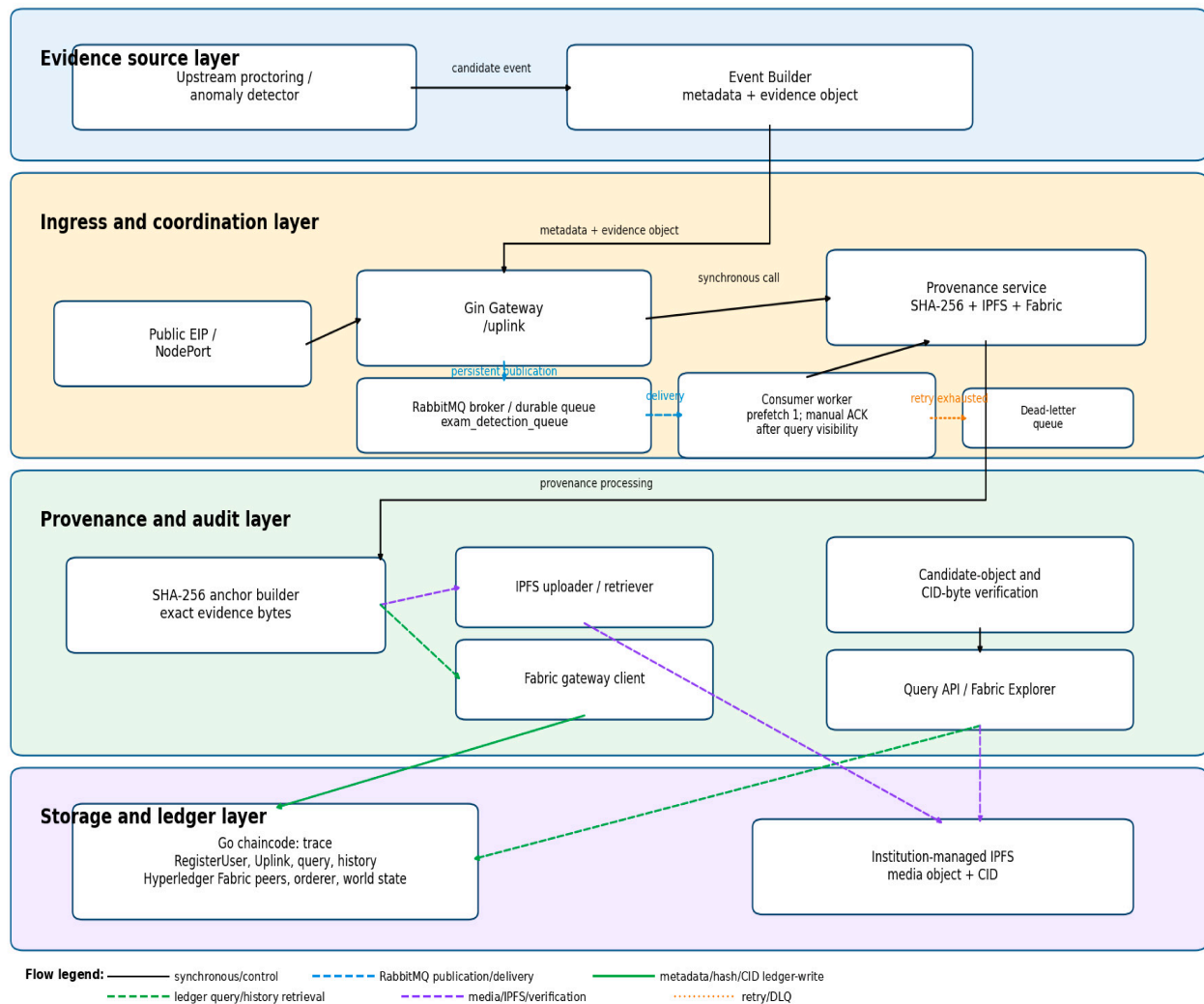
Two mechanisms define the research boundary. The first is an integrity-by-reference provenance mechanism that records compact event semantics and a SHA-256 evidence anchor on-chain while retaining the evidence object off-chain. The second is a queue-mediated ingress mechanism that separates client-facing persistent-message publication from downstream completion and exposes both states to measurement. The detection component is used as a modular upstream event source.

Let  $e$  denote a suspicious examination event, let  $b(e)$  denote the exact evidence byte sequence submitted for retention, and let  $c(e)$  denote the IPFS content identifier. The chain record stores `imgHash` as  $h(e) = \text{SHA-256}(b(e))$  and `ipfsCid` as  $c(e)$ . A supplied candidate object is verified by hashing its exact byte sequence and comparing the result with  $h(e)$ . CID-level byte audit retrieves the object referenced by  $c(e)$ , hashes the retrieved bytes, and compares the result with the same anchor.

The analytical framework links three system boundaries. At the storage boundary, larger ledger payloads increase transaction-path work, whereas compact anchors keep ledger state independent of media-object size. At the write boundary, sharp latency inflation and failure accumulation identify the transition from an operational region to saturation. At the ingress boundary, persistent broker publication shortens the client-facing request path and exposes excess arrivals as queue backlog, while consumer-enabled processing measures the time and outcomes required to convert admitted tasks into object-retained, ledger-committed, query-visible, and acknowledged evidence. These expectations guide RQ1–RQ3 and the interpretation of completion, latency, success, admission, and queue-drain metrics.

#### 3.2. Software Architecture Description and Implementation Views

Figure 1 presents the implemented software architecture of the media-evidence provenance pipeline. The view-based description follows software-architecture description practice and separates use-case, logical, process, deployment, and data-flow concerns so that the figure maps to the executed software components [23].



**Figure 1.** Implemented software architecture of the media-evidence provenance pipeline.

The use-case view contains three actors: an upstream event producer that forms suspicious-event packages, a gateway that accepts evidence submissions, and an authorized reviewer that later queries or verifies evidence. The logical view separates the evidence source layer, the ingress and coordination layer, the synchronous provenance service layer, the storage and ledger layer, and the audit/review layer. This decomposition keeps detector output, ledger storage, queue publication, and evidence verification as distinct system concerns.

The process view separates three executed paths: synchronous provenance submission, producer-only RabbitMQ publication, and consumer-enabled completion. In the synchronous path, the client-facing request waits for SHA-256 generation, IPFS retention, and Fabric anchoring. In the producer-side asynchronous path, the gateway publishes a persistent message to the durable queue exam\_detection\_queue and returns HTTP 202 after the publish call completes without error; the PTS and JMeter spike runs used this topology with no active consumer. In the consumer-enabled path, one worker retrieves a queued message, invokes the provenance service, and acknowledges the message only after IPFS retention, successful Fabric commit, and query visibility; exhausted retries are routed to the dead-letter queue. The deployment view maps the gateway, RabbitMQ, and consumer to the Tencent Kubernetes Engine service tier and the Fabric/IPFS provenance substrate to a separate cloud host.

The data-flow view is shown explicitly in Figure 1. Solid black arrows denote synchronous/control flow; blue dashed arrows, RabbitMQ publication/delivery; solid green arrows, metadata/hash/CID ledger-write flow; green dashed arrows, ledger query/history retrieval; purple dashed arrows, media/IPFS or verification-support flow; and orange dotted arrows, retry/dead-letter routing.

### 3.3. Upstream Evidence Generation Service

The upstream service converts detection output into a provenance-ready event package containing the examination identifier, examinee identifier, timestamp, detector output string, event type, original file name, and evidence object. The provenance service subsequently computes the SHA-256 anchor, retains the object in IPFS, and binds the resulting CID to the Fabric record. This event structure is compatible with the upstream detector pipeline.

The upstream detector is used as a modular producer of evidence-bearing events. The event source follows a detector-oriented prototype previously reported by the authors [24], while the present study follows the evidence through semantic binding, off-chain retention, candidate verification, and burst admission.

### 3.4. Hybrid On/Off-Chain Provenance Anchoring

#### 3.4.1. Metadata Schema and Evidence Binding

The provenance layer stores compact, integrity-critical metadata on-chain and retains evidence objects off-chain. The chain transaction records the event semantics,  $c(e)$ , and  $h(e)$ , enabling candidate-object verification and CID-level byte audit through one SHA-256 anchor. This integrity-by-reference design follows content-addressed storage [9], permissioned blockchain governance [10], Fabric transaction-path evidence [11], decentralized object retention [12], IPFS-based evidence preservation [13], judicial evidence preservation [14], and document-version provenance [15]. The Caliper hybrid workload measures the Fabric transaction carrying bounded metadata,  $h(e)$ , and  $c(e)$ ; IPFS upload and CID retrieval verification are completed during run preparation and remain outside the Caliper transaction interval.

The key design principle is that verifiability must survive storage separation. The ledger stores both the identifier and the semantics needed to reconstruct the event during later review. In the prototype, the event record captures the examination context, detector result, evidence reference, and hash anchor. This allows authorized reviewers to retrieve both the event description and the object reference from a single provenance query.

Candidate-object verification and retained-object audit apply the same SHA-256 comparison to different byte sources. The candidate path hashes a supplied object, whereas the CID path hashes the exact bytes retrieved from  $c(e)$ . A match binds the inspected bytes to the evidence anchor recorded in Fabric.

In the evaluated prototype, IPFS was deployed as an institution-managed content-addressed retention node co-located with the provenance substrate. The ledger-path payload-placement runs prepared and verified the IPFS object before the measured Caliper round, thereby isolating the Fabric transaction-path cost of compact anchoring. Operational deployments can add encrypted retention, institution-controlled pinning or IPFS Cluster policies, independent backup, and periodic CID retrieval probes.

#### 3.4.2. Chaincode Design and Access Semantics

The provenance contract is implemented as Go chaincode using the Hyperledger Fabric Contract API and is deployed under the contract name trace. Its evidence-domain interface comprises RegisterUser, Uplink, GetUserInfo, GetEvidenceInfo, GetAllEvidenceInfo, GetEvidenceList, and GetEvidenceHistory. These operations record submitting

identities and declared institutional roles, write evidence records, retrieve individual or range-based records, inspect user-associated evidence, and query Fabric key history.

The ledger schema records evidenceId, examId, studentId, timestamp, yoloResults, eventType, imgHash, storageMode, mediaPayloadBase64, ipfsCid, originalFileName, submitterId, transactionId, and ledgerTimestamp. The direct large-payload and compact-anchor ledger conditions use the same Uplink argument order. The direct condition supplies the Base64 media field and leaves ipfsCid empty; the compact-anchor condition leaves the media field empty and supplies imgHash, ipfsCid, and the original file name.

The ledger-path payload-placement experiment stresses the Uplink transaction. In the direct large-payload condition, Uplink writes metadata together with the large Base64 field into Fabric state. In the compact-anchor condition, the gateway commits bounded metadata, the candidate-verification value, and the IPFS CID, while the media body is retained outside the ledger path. Query and history functions then reconstruct the anchored record for audit without re-submitting media bytes.

Candidate-object verification and CID-level byte audit are service-layer procedures that query the anchored imgHash and recompute SHA-256 over the inspected bytes. The chaincode provides the tamper-evident evidence-write, query, and history interface stressed by the Caliper runs, whereas verification decisions are executed at the service layer.

### 3.4.3. Threat Model and Security Assumptions

The prototype operates within an institution-controlled permissioned-ledger environment. Fabric history, SHA-256 verification, and separated application endpoints constitute the evaluated controls. Operational deployments add application-endpoint authentication and fine-grained authorization. The threat boundary covers five operational risks: modified or replayed client submissions, unauthorized observation of off-chain references or objects, interrupted submission before anchoring, privileged misuse of audit or retention interfaces, and inconsistency between off-chain retention and on-chain provenance.

Within this boundary, Fabric transaction history makes committed metadata changes traceable, SHA-256 recomputation detects byte substitution, and gateway-mediated endpoints separate submission, query, verification, and audit operations. The institution-managed IPFS node provided object availability during the experiments, while replicated retention and recovery are treated as deployment-scale controls.

Deployment-scale security and recovery controls include mTLS or signed-request authentication, encrypted retention, HSM- or KMS-backed signing-key management, IPFS pinning or replication, idempotency keys, durable retry records, dead-letter queues, and multi-party approval for high-impact evidence release or retention-policy changes.

### 3.4.4. Executable Candidate-Object Verification

For a queried evidence identifier, the service reads  $h(e)$  and  $c(e)$  from Fabric. Candidate-object verification hashes the exact byte sequence of a supplied object and compares the result with  $h(e)$ . CID-level byte audit retrieves the object referenced by  $c(e)$ , hashes the retrieved byte sequence, and applies the same comparison. Matching values establish consistency between the inspected bytes and the anchored evidence value.

The two procedures verify different objects: the candidate path evaluates an externally supplied object, whereas the CID path evaluates the bytes retained under the recorded IPFS reference. Both procedures are byte-identity checks; transcoding, metadata alteration, or any other byte-level change produces a different digest.

Let  $B(o)$  denote the exact byte sequence extracted from object  $o$  and let  $h(e)$  be the SHA-256 value stored in the chain record. The candidate-object decision is:

$$V(e, \hat{o}(e)) = 1 \text{ if } \text{SHA-256}(B(\hat{o}(e))) = h(e), \text{ and } V(e, \hat{o}(e)) = 0 \text{ otherwise.} \quad (1)$$

CID-level byte audit applies Equation (1) to the byte sequence returned by  $\text{Fetch}(c(e))$ .

### 3.4.5. Algorithmic Procedures

CID-level byte audit retrieves  $\text{Fetch}(c(e))$ , computes SHA-256 over the retrieved bytes, and compares the result with  $h(e)$ .

Complexity analysis: Algorithm 1 hashes and uploads the evidence bytes in  $O(|b(e)|)$  time. The Fabric payload contains a constant number of schema fields and excludes the media body; its size is bounded by metadata-field limits rather than by  $|b(e)|$ . Algorithm 2 is linear in the candidate-object byte sequence plus one ledger query. CID-level audit adds retrieval and hashing in  $O(|b(e)|)$  time. Table 2 summarizes the principal evidence fields, contract operations, and audit roles.

---

#### Algorithm 1 Hybrid Evidence Anchoring Procedure

---

Input: suspicious event  $e$ , raw evidence object  $o(e)$ , metadata tuple  $m(e)$

Output: anchored record  $\rho(e)$

1. Obtain the exact evidence byte sequence  $b(e)$  from object  $o(e)$ .
  2. Compute the evidence anchor  $h(e) = \text{SHA-256}(b(e))$ .
  3. Upload  $b(e)$  to IPFS and obtain the content identifier  $c(e)$ .
  4. Construct compact on-chain payload  $p(e) = \langle m(e), h(e), c(e) \rangle$ .
  5. Submit  $p(e)$  to chaincode and obtain transaction record  $tx(e)$ .
  6. Set  $a(e) = \langle tx(e) \rangle$  and record the association among the evidence identifier,  $tx(e)$ , and  $c(e)$ .
  7. Return  $\rho(e) = \langle m(e), h(e), c(e), a(e) \rangle$ .
- 

---

#### Algorithm 2 Candidate-Object Verification Procedure

---

Input: evidence identifier  $id(e)$ , verification candidate  $\hat{o}(e)$

Output: verification decision  $V(e, \hat{o}(e))$

1. Query the chain record and read  $\langle h(e), c(e), a(e) \rangle$  for  $id(e)$ .
  2. Obtain the exact byte sequence  $B(\hat{o}(e))$  from the verification candidate.
  3. Compute  $\hat{h}(e) = \text{SHA-256}(B(\hat{o}(e)))$  and set  $V(e, \hat{o}(e)) = 1$  when  $\hat{h}(e) = h(e)$ ; otherwise set  $V(e, \hat{o}(e)) = 0$ .
  4. Return the verification decision together with the queried provenance context.
- 

**Table 2.** Core evidence fields, provenance operations, and audit purposes.

| Field/Operation | Meaning in the Prototype                                      | Role in Provenance/Audit                                          |
|-----------------|---------------------------------------------------------------|-------------------------------------------------------------------|
| examId          | Unique identifier of the examination session                  | Binds evidence to a concrete assessment context                   |
| studentId       | Unique identifier of the examinee linked to the event         | Supports accountable review and case-level query                  |
| timestamp       | Time at which the suspicious event was recorded               | Supports temporal ordering, replay analysis, and event-time audit |
| yoloResults     | Detector output string from the upstream service              | Preserves event semantics beyond raw image storage                |
| imgHash         | SHA-256 value computed over the exact evidence byte sequence  | Supports candidate-object verification and CID-level byte audit   |
| ipfsCid         | Off-chain content identifier for the retained evidence object | Supports retrieval without forcing object-level on-chain storage  |

Table 2. Cont.

| Field/Operation               | Meaning in the Prototype                                                                                          | Role in Provenance/Audit                                                               |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| Evidence uplink               | Writes compact metadata and integrity references to Fabric                                                        | Creates the tamper-evident provenance anchor                                           |
| Evidence query                | Retrieves anchored metadata for review                                                                            | Supports case review, provenance inspection, and verification-context retrieval.       |
| Candidate-object verification | Hashes the supplied candidate bytes and compares the result with imgHash                                          | Verifies consistency of an externally supplied object with the anchored evidence value |
| CID-level byte audit          | Retrieves the object referenced by ipfsCid, hashes the exact returned bytes, and compares the result with imgHash | Verifies the retained byte sequence against the anchored evidence value                |

### 3.5. Queue-Mediated Ingress Coordination

The prototype supports synchronous provenance submission and asynchronous queue-mediated submission. In synchronous mode, the evidence-submission request waits for provenance processing. In asynchronous mode, the gateway publishes the request as a persistent RabbitMQ message and returns HTTP 202 after the publish call completes without error. The burst experiments used a producer-only topology in which published messages accumulated as ready messages. The consumer-enabled completion experiment extended the asynchronous path through message delivery, IPFS retention, Fabric commit confirmation, query visibility, and manual acknowledgement.

This distinction is especially important under bounded cloud resources. Let  $A(t, \Delta t)$  denote arrivals during interval  $\Delta t$  and let  $S(t, \Delta t)$  denote completed service during the same interval. The backlog dynamics can then be represented as:

$$Q(t + \Delta t) = \max\{0, Q(t) + A(t, \Delta t) - S(t, \Delta t)\} \quad (2)$$

When arrivals exceed the rate at which a subsequent consumer can drain the queue,  $Q(t)$  increases and backlog becomes visible at the broker. The PTS and JMeter experiments measure the producer path from HTTP receipt to persistent queue publication. The consumer-enabled experiment measures queue drain and admission-to-completion under an active consumer. Caliper separately characterizes the Fabric ledger path.

#### 3.5.1. Producer-Side RabbitMQ Publication Configuration

RabbitMQ was deployed as a single-replica Kubernetes StatefulSet using rabbitmq:3.8.9-management-alpine. The queue exam\_detection\_queue was declared durable, and messages were published with persistent delivery mode. The asynchronous gateway was the sole producer and returned HTTP 202 after the RabbitMQ publish call completed without error.

The producer-side spike topology contained one gateway producer and zero consumers. RabbitMQ QoS/prefetch remained unset, and ready messages accumulated during the measurement interval. The durable queue was inspected and purged before each repeated producer-side run.

HPA was disabled and the asynchronous gateway, RabbitMQ replica count, queue declaration, message-delivery mode, request schema, and public NodePort endpoint were held constant across repeated runs. Runtime queue observations recorded ready-message accumulation and zero active consumers during the spike window.

This configuration measures gateway-to-queue publication and ready-message accumulation. It is complemented by the consumer-enabled protocol in Section 3.5.2, which

measures IPFS retention, Fabric commit, query visibility, acknowledgement, retry, and dead-letter outcomes.

### 3.5.2. Consumer-Enabled Completion Protocol

The consumer-enabled completion experiment comprised three independent runs of 30 messages using the deployed RabbitMQ 3.12.14 broker. Each message carried a 384,000-byte media payload. One RabbitMQ consumer used manual acknowledgement, a prefetch count of one, and a maximum of three retry publications. The source message was acknowledged only after the object was retained on the managed IPFS node, the Fabric transaction returned successful commit status, and a subsequent query exposed the committed record; exhausted retries were routed to a dedicated dead-letter queue.

Producer and consumer workloads were pinned to two TKE workers with identical 4-vCPU/8-GB specifications. The producer ran on node 10.0.16.17, while the consumer, asynchronous gateway, and RabbitMQ ran on node 10.0.16.11; the consumer invoked the provenance service on the Fabric/IPFS host. A newly registered test user was used for each run so that the user-associated ledger list began empty. The main queue and dead-letter queue were purged before each run.

Per-message timestamps recorded client request start, HTTP 202 acceptance, consumer receive, IPFS start and end, Fabric start and end, query visibility, and final acknowledgement. Queue-drain time was measured from the final HTTP 202 response to the final completed acknowledgement, and total completion time from the first request to the final acknowledgement. The protocol also counted accepted and completed messages, retry publications, producer failures, dead-letter outcomes, and missing stage timestamps.

## 3.6. Experimental Environment and Protocol

All experiments were executed on a segmented Tencent Cloud deployment. Hyperledger Fabric 2.5, IPFS, and the Caliper workspace ran on one dedicated lightweight CVM configured with 4 vCPUs, 8 GB RAM, a 60 GB SSD system disk, and a 5 Mbps public-bandwidth allocation. Fabric Explorer showed channel mychannel, two peer endpoints, and one orderer endpoint within this single-host provenance substrate. The application tier ran in a Tencent Kubernetes Engine cluster in Guangzhou and hosted the Gin gateways, RabbitMQ, monitoring components, and the event-generation service. For the consumer-enabled experiment, producer and consumer workloads were pinned to two TKE workers with identical 4-vCPU/8-GB specifications. Fixed resources were used throughout the reported comparisons.

The benchmark paths were segmented by measurement boundary. JMeter and the retained Tencent PTS tasks accessed the TKE gateway through public EIP/NodePort endpoints. Communication between the asynchronous gateway and RabbitMQ used Kubernetes service networking. Caliper executed on the Fabric/IPFS host and exercised the local containerized ledger path. In the consumer-enabled experiment, the producer reached the asynchronous gateway through the TKE NodePort, and the consumer invoked the provenance service on the Fabric/IPFS host through its configured public backend endpoint. The 5 Mbps allocation therefore applied to the provenance-host segment of the completion path. Raw video streams remained outside the message queue.

### 3.6.1. Deployment and Workload-Generation Details

JMeter 5.6.3 was launched from a Windows host outside the TKE cluster and targeted the public/uplink NodePort endpoint. The fixed-concurrency calibration run used 150 users, a 1 s ramp-up, and a 180 s duration with HPA disabled and the event-generation service fixed to one replica. Requests used application/x-www-form-urlencoded fields of

student\_id, exam\_id, and image\_base64. Deployment checks used either a fixed Base64 placeholder or CSV-driven screen-capture samples.

For the spike-ingress comparison, JMeter fixed the submitted request count with 1000 users, 0 s ramp-up, one loop, and a Synchronizing Timer. Tencent PTS used 1000 virtual users over 60 s against the public gateway endpoint and generated repeated client loops under a closed-loop load model. The service tier routed traffic either to the synchronous gateway or to the RabbitMQ producer endpoint. The consumer-enabled workload followed the fixed protocol in Section 3.5.2. Raw PTS reports, JMeter files, RabbitMQ observations, consumer logs, per-message timing records, and Fabric/IPFS logs were retained.

### 3.6.2. Replication-Oriented Implementation Notes

The producer-side ingress configuration used a single RabbitMQ 3.8.9 broker, the durable queue exam\_detection\_queue, persistent messages, one producer gateway, zero consumers, and unset QoS/prefetch. The synchronous gateway was exposed on NodePort 30264 and the RabbitMQ producer endpoint on NodePort 31846; both served/uplink with the same request schema. The queue was inspected and purged before repeated producer-side runs. The consumer-enabled experiment used the deployed RabbitMQ 3.12.14 broker and the fixed protocol in Section 3.5.2. The two experiment sets are reported independently: the former measures HTTP-to-queue publication, whereas the latter measures queue drain and downstream completion. Each completion run began with empty main and dead-letter queues and a newly registered user with an initially empty user-associated ledger list.

The deployed Fabric network used channel mychannel and Go chaincode for evidence anchoring and query operations, with two visible peer endpoints and one orderer endpoint. Endorsement, ordering, block cutting, timeout, state database, and contract interface remained constant across the ledger comparisons. Ledger payload placement, payload size, offered load, and ingress mode formed the experimental factors.

The verification anchor was instantiated as SHA-256 over the exact evidence byte sequence. Both candidate-object verification and CID-level byte audit recompute this value over the bytes being inspected.

### 3.6.3. Ethics and Demonstration-Data Note

The prototype evaluation used synthetic demonstration records and system-generated benchmark payloads without examinee or invigilator participation. Any identifiers shown in workflow or verification illustrations are synthetic and serve only to demonstrate system operation.

### 3.6.4. Metric Semantics Used in This Article

Observed send rate denotes the rate recorded by the Caliper controller during a benchmark round. Caliper completion rate denotes terminal transaction completions within the ledger measurement window and is reported separately from successful tx/s. Ledger success rate is the fraction of successful outcomes among terminal outcomes reported for a round. API-level RPS and recorded client loops denote Tencent PTS observations under the closed-loop load model. An asynchronous HTTP 202 response denotes that the RabbitMQ publish call completed without error. In the consumer-enabled experiment, queue-drain time is the interval from the final HTTP 202 response to the final successful acknowledgement, while total completion time is the interval from the first request to the final acknowledgement. Backend IPFS stage time and Fabric commit time are derived from service timestamps; completion requires successful commit status and subsequent query visibility. Caliper measurements characterize Fabric ledger-path behavior. For the repeated 20-TPS ledger-path runs, effective completion time is calculated as the terminal transaction

count divided by the run-level Caliper completion rate, distinguishing the 60 s fixed-rate submission phase from the time required to reach terminal outcomes.

Tables 3 and 4 report on the deployment environment and the experimental protocol.

**Table 3.** Experimental environment and evaluation setup.

| Item                             | Configuration/Description                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Cloud deployment                 | Segmented Tencent Cloud prototype with a Fabric/IPFS/Caliper host and a separate TKE API/RabbitMQ tier in Guangzhou.                                                                                                                                                                                                                                                                                                   |
| Fabric/IPFS host                 | 4 vCPU, 8 GB RAM, 60 GB SSD system disk, and 5 Mbps public-bandwidth allocation.                                                                                                                                                                                                                                                                                                                                       |
| Measured network paths           | JMeter and retained PTS tasks targeted public EIP/NodePort gateway endpoints; gateway-to-RabbitMQ communication used Kubernetes service networking; Caliper exercised the local containerized Fabric path.                                                                                                                                                                                                             |
| Ledger platform                  | Hyperledger Fabric 2.5; channel mychannel; two visible peer endpoints and one orderer endpoint on the single provenance host.                                                                                                                                                                                                                                                                                          |
| Off-chain store                  | Institution-managed IPFS node co-located with Fabric; hybrid runs prepared the object and verified CID retrieval before the measured Caliper round.                                                                                                                                                                                                                                                                    |
| TKE service tier                 | Gin synchronous gateway, RabbitMQ producer endpoint, RabbitMQ broker, monitoring components, and one event-generation-service replica; HPA disabled.                                                                                                                                                                                                                                                                   |
| Chaincode                        | Go Fabric Contract API; contract trace with RegisterUser, Uplink, GetUserInfo, GetEvidenceInfo, GetAllEvidenceInfo, GetEvidenceList, and GetEvidenceHistory.                                                                                                                                                                                                                                                           |
| RabbitMQ                         | Producer-side spike runs: rabbitmq:3.8.9-management-alpine; one-replica StatefulSet; AMQP 5672; durable exam_detection_queue; persistent messages; producer-only measured path. Consumer-enabled completion runs: RabbitMQ 3.12.14; one manual-acknowledgement consumer; prefetch 1; maximum of three retries; dedicated dead-letter queue. The two configurations are reported under separate measurement boundaries. |
| Gateway exposure                 | Synchronous NodePort 30264 and RabbitMQ producer NodePort 31846; both expose/uplink with the same-form schema.                                                                                                                                                                                                                                                                                                         |
| Benchmark tools                  | Hyperledger Caliper for ledger-path tests; Tencent PTS for repeated 1000-VU closed-loop tests; JMeter 5.6.3 for calibration and synchronized fixed-request replay.                                                                                                                                                                                                                                                     |
| Monitoring and retained evidence | Caliper reports, PTS exports, JMeter JMX/CSV files, RabbitMQ observations, Kubernetes manifests, chaincode, API schemas, and Fabric/IPFS logs.                                                                                                                                                                                                                                                                         |

**Table 4.** Fixed prototype parameters and replication boundaries.

| Component/Protocol          | Fixed Setting                                                                                                                               | Replication Meaning                                                            |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| Caliper ledger-path workers | 5 workers; 20 prepared users per worker; user registration completed before each measured round.                                            | Keeps worker and identity preparation identical for the two ledger conditions. |
| Low-load ledger-path runs   | 5 independent runs; 300 measured transactions; fixed-rate target 5 TPS.                                                                     | Establishes matched completion at low load.                                    |
| Capacity-separation runs    | 3 independent runs; 60 s; fixed-rate target 20 TPS.                                                                                         | Exposes ledger-path divergence under the same observed send rate.              |
| Direct large-payload        | 384,000 decoded bytes; 512,000 Base64 characters; SHA-256 fcb3b88d13357fa7b8189bc8778287373d8aa43fcd0c6ef5346e1b99e41c164.                  | Carries the media-sized field through Fabric.                                  |
| Compact-anchor payload      | Bounded metadata, the same SHA-256 anchor, and a real IPFS CID; media field empty.                                                          | Measures compact ledger anchoring after IPFS preparation.                      |
| Ledger clean-state protocol | Fabric/IPFS prototype restarted; prepared users registered before the measured Caliper round and excluded from reported transaction counts. | Reduces cross-run state contamination in ledger-path comparisons.              |

Table 4. *Cont.*

| Component/Protocol                 | Fixed Setting                                                                                                                                                                                                                                     | Replication Meaning                                                                                                                           |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Ingress clean-state protocol       | RabbitMQ queue inspected and purged before each repeated asynchronous PTS/JMeter run; the main queue and dead-letter queue were purged before each consumer-enabled completion run, and a newly registered user was used for each completion run. | Aligns producer-side spike runs with an empty initial queue and consumer-enabled runs with equivalent queue and user-associated ledger state. |
| Producer-side RabbitMQ publication | One broker; durable queue; persistent messages; gateway-only producer; HTTP 202 after the RabbitMQ publish call completed without error; zero consumers; QoS/prefetch unset.                                                                      | Defines the PTS/JMeter producer-side measurement boundary.                                                                                    |
| Tencent PTS                        | 3 runs per mode; 1000 VUs; 60 s; closed-loop execution against the public gateway endpoint.                                                                                                                                                       | Reports observed client loops, RPS, HTTP outcomes, and latency.                                                                               |
| JMeter spike                       | 3 runs per mode; 1000 users; 0 s ramp-up; one loop; Synchronizing Timer 1000/30,000 ms.                                                                                                                                                           | Provides fixed-request completion and latency evidence.                                                                                       |
| Consumer-enabled completion        | 3 independent runs; 30 messages per run; 384,000-byte payload; one consumer; prefetch 1; manual acknowledgement after IPFS retention, successful Fabric commit, and query visibility; maximum of three retries; dedicated DLQ.                    | Measures admitted-message completion through IPFS retention, ledger confirmation, query visibility, and acknowledgement.                      |
| Completion placement and state     | Producer pinned to TKE node 10.0.16.17; consumer, gateway, and RabbitMQ pinned to same-spec node 10.0.16.11; fresh registered user per run; main queue and DLQ purged before each run.                                                            | Maintains equivalent worker resources, routing, queue state, and initial user-associated ledger state.                                        |
| Completion metrics                 | Queue drain; IPFS storage; Fabric commit; accepted/completed messages; retry publications; producer failures; DLQ outcomes; first request to final acknowledgement.                                                                               | Separates HTTP admission from queue drain and downstream completion.                                                                          |
| Verification                       | SHA-256 over exact bytes; candidate verification and CID retrieval audit compare with imgHash.                                                                                                                                                    | Uses one anchor for two object-specific checks.                                                                                               |
| Scale-out variables                | Multiple consumers, varied prefetch, fault-injected retry and DLQ handling, worker drain, multi-host Fabric, and replicated object retention.                                                                                                     | Defines the parameter space for deployment-scale evaluation beyond the measured one-consumer completion experiment.                           |

### 3.6.5. Payload and Evidence-Boundary Profile

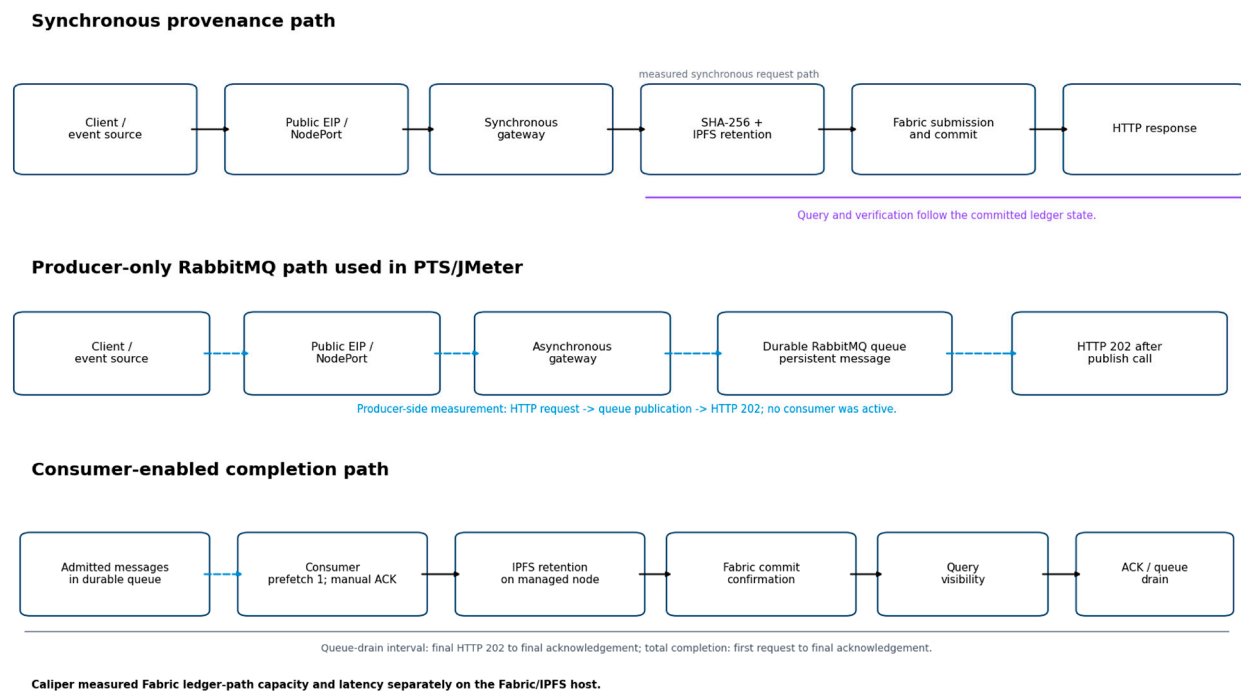
Payload design was specified separately for ledger and ingress measurements. The direct Caliper path carried a deterministic synthetic media field of 384,000 decoded bytes, producing 512,000 Base64 characters. The hybrid Caliper path carried bounded evidence metadata, the SHA-256 anchor, and a real IPFS CID; IPFS upload and retrieval verification were completed before the measured ledger round. PTS and JMeter used the same application/x-www-form-urlencoded API schema with fixed placeholder or CSV-driven screen-capture samples. The consumer-enabled experiment used the same request schema and a 384,000-byte media payload; the remaining completion parameters are defined in Section 3.5.2. These workloads provide controlled boundary conditions for the reported comparisons.

## 4. Results and Analysis

The results are organized around the three research questions. Sections 4.1 and 4.2 validate pipeline operation and candidate-object verification. Section 4.3 identifies the provenance-write operating region. Section 4.4 compares direct large-payload ledger submission with compact anchoring. Sections 4.5 and 4.6 report producer-side ingress behavior, consumer-enabled queue drain and completion, and the interim findings.

#### 4.1. Pipeline Implementation Validation

Figure 2 presents the three runtime paths used in the evaluation. The synchronous request traverses the gateway, SHA-256 generation, IPFS retention, and Fabric submission and commit before the HTTP response; query and verification access follow the committed ledger state. The producer-side RabbitMQ path covers the public gateway, persistent-message publication to exam\_detection\_queue, and the HTTP 202 response. The consumer-enabled path begins with an admitted message in the durable queue and continues through delivery, IPFS retention, Fabric commit, query visibility, and manual acknowledgement. Ready messages accumulated during the producer-only spike runs after the RabbitMQ publish call completed without error, whereas the consumer-enabled experiment followed the queue until all admitted messages were acknowledged. Caliper measured ledger-path capacity and latency separately on the Fabric/IPFS host.



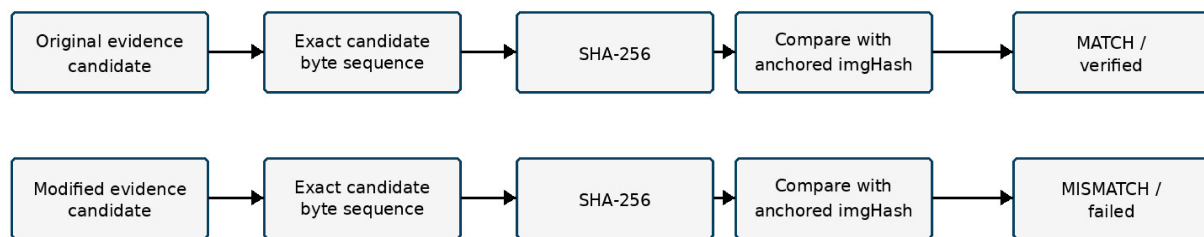
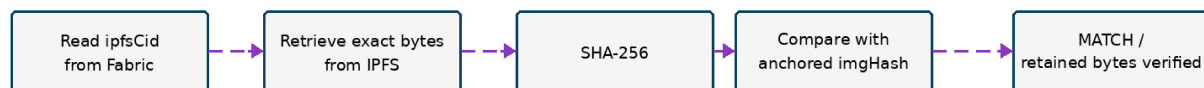
**Figure 2.** Runtime paths and measurement boundaries for synchronous submission, producer-only RabbitMQ publication, and consumer-enabled completion.

#### 4.2. Candidate-Object Verification

Figure 3 and Table 5 report two SHA-256 verification checks. The original candidate produced the anchored value, whereas a one-byte modification produced a different digest. The object retrieved through the recorded IPFS CID reproduced the anchored value.

**Table 5.** Candidate-object verification and CID-level byte-audit cases.

| Case                          | Evidence ID        | Anchored SHA-256 (Abbreviated) | Recomputed SHA-256 (Abbreviated) | Outcome                       |
|-------------------------------|--------------------|--------------------------------|----------------------------------|-------------------------------|
| Original evidence candidate   | 029769282891550720 | fcba3b88...e41c164             | fcba3b88...e41c164               | Match/verified                |
| One-byte modified candidate   | 029769282891550720 | fcba3b88...e41c164             | 8374aa9e...d5e345                | Mismatch/failed               |
| Object retrieved from ipfsCid | 029769282891550720 | fcba3b88...e41c164             | fcba3b88...e41c164               | Match/retained bytes verified |

**Candidate-object verification****CID-level retained-object byte audit**

Both procedures use the same SHA-256 anchor but verify different objects: a supplied candidate and the bytes retrieved from the recorded CID.

**Figure 3.** Candidate-object verification and CID-level byte-audit workflows. Solid arrows denote the candidate-object verification flow, whereas purple dashed arrows denote CID-based retrieval and retained-object byte-audit flow.

Table 5 abbreviates the displayed SHA-256 values; full synthetic values are provided in the Supplementary Materials.

#### 4.3. Provenance-Substrate Read/Write Boundary

As shown in Table 6 and Figure 4, the provenance-substrate write and read paths were evaluated across a load sweep. In the low-to-moderate region, the evidence-uplink write path tracked the configured load fairly closely: an observed completion rate of 29.1 TPS at an observed send rate of 30.1 TPS, 48.4 TPS at 50.1 TPS, and 96.2 TPS at 100.0 TPS. Mean latency remained below half a second through the 100-TPS point. The slight decrease in mean latency from 30 TPS (0.18 s) to 50 TPS (0.15 s) may reflect Hyperledger Fabric block-cutting dynamics, because a moderately higher arrival rate can reach the block-size threshold before the timeout threshold. In the higher-load rounds, the benchmark client recorded observed send rates below the nominal controller targets. Once the configured load increased further, latency and failure counts rose sharply: the 150-TPS fixed-rate run produced an observed completion rate of 145.8 TPS but required 8.02 s average latency, while the 200-, 250-, and 300-TPS runs remained heavily stressed and continued to exhibit multi-second average delay. These higher-load rounds identify the practical operating boundary of the single-host provenance prototype. Panel (a) maps each nominal controller target to its observed send rate, while panels (b)–(d) use observed send TPS as the horizontal coordinate for successful throughput, Caliper completion, ledger success, and mean latency.

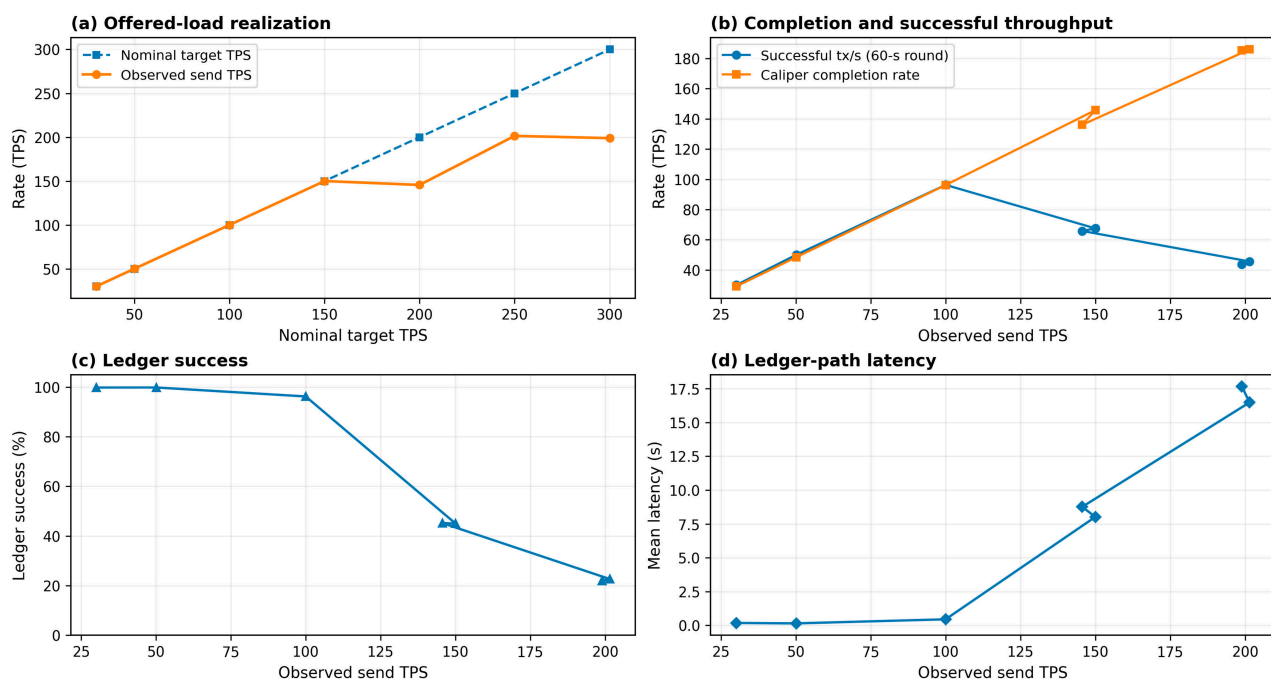
The read path was substantially lighter. A representative high-load query round characterizes this path because metadata retrieval bypasses the ordering-and-commit pipeline that dominates write-side execution. The evidence-query benchmark sustained 293.9 TPS with 0.01 s average latency and no failures. This asymmetry localizes the principal performance constraint to the provenance write path. Under the evaluated topology, evidence submission and ledger-path completion define the measured service boundary, while metadata retrieval remains lightweight.

The saturation rows above 150 TPS represent write-boundary stress. Under the fixed topology, the likely bottleneck chain includes proposal execution and endorsement at the peer boundary, ordering and block cutting, validation and commit, gateway waiting, and client-side backlog when requests outpace commit completion. IPFS object handling can

add delay when media-bearing objects are retained; the light query row and the ledger-path payload-placement contrast place the dominant overload path at the provenance write boundary. Multi-peer and multi-orderer experiments can separate peer endorsement, orderer batching, and commit validation more finely.

**Table 6.** Provenance-substrate read/write load sweep.

| Operation      | Target TPS | Observed Send TPS | Terminal Success/Failure | Ledger Success (%) | Successful Tx/s (60 s round) | Caliper Completion Rate (TPS) | Avg./Max Latency (s) |
|----------------|------------|-------------------|--------------------------|--------------------|------------------------------|-------------------------------|----------------------|
| Write (uplink) | 30         | 30.1              | 1801/4                   | 99.78              | 30.0                         | 29.1                          | 0.18/2.07            |
| Write (uplink) | 50         | 50.1              | 2999/6                   | 99.80              | 50.0                         | 48.4                          | 0.15/2.07            |
| Write (uplink) | 100        | 100.0             | 5776/229                 | 96.19              | 96.3                         | 96.2                          | 0.45/2.46            |
| Write (uplink) | 150        | 150.0             | 4051/4952                | 45.00              | 67.5                         | 145.8                         | 8.02/17.05           |
| Write (uplink) | 200        | 145.6             | 3949/4792                | 45.18              | 65.8                         | 136.1                         | 8.76/22.05           |
| Write (uplink) | 250        | 201.5             | 2738/9353                | 22.64              | 45.6                         | 186.0                         | 16.48/42.94          |
| Write (uplink) | 300        | 198.9             | 2621/9322                | 21.95              | 43.7                         | 185.5                         | 17.66/43.12          |
| Read (query)   | 300        | 300.0             | 17,578/0                 | 100.00             | 293.0                        | 293.9                         | 0.01/0.35            |



**Figure 4.** Provenance write-path load sweep. (a) Nominal target TPS and observed send TPS; (b) successful tx/s and Caliper completion rate versus observed send TPS; (c) ledger success versus observed send TPS; (d) mean latency versus observed send TPS.

The 30- and 50-TPS rounds define the stable operating region. The 100-TPS round marks an upper transition point: mean latency remained 0.45 s, but ledger success declined to 96.19%. Above 100 TPS, failure accumulation and multi-second latency make the rounds unsuitable for integrity-sensitive evidence submission. The deployable region is therefore identified from successful write throughput, ledger success rate, and latency together.

The success-normalized column in Table 6 makes this operating boundary explicit. Successful write throughput remains close to the offered load through 100 TPS, then drops to 67.5, 65.8, 45.6, and 43.7 tx/s at the 150-, 200-, 250-, and 300-TPS saturation targets. Succ. tx/s is computed from successful terminal transactions over the nominal 60 s round,

whereas Caliper TPS is the completion rate reported within Caliper’s measurement window. The table therefore separates framework-reported completion rate from deployable successful write throughput.

#### 4.4. Ledger-Path Payload-Placement Evaluation

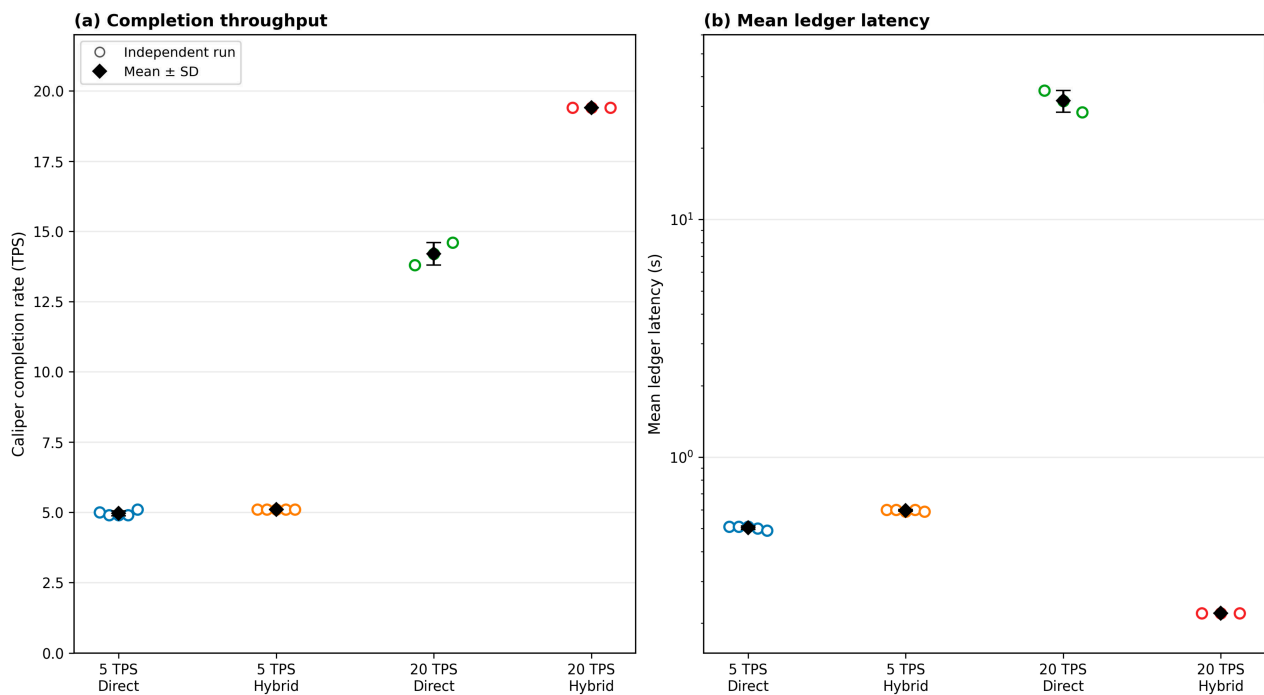
As shown in Table 7 and Figure 5, the ledger-path payload-placement experiment used two matched Fabric transaction conditions. Five independent 300-transaction runs at 5 TPS established a low-load baseline, and three independent 60 s runs at 20 TPS provided a capacity-separation point. Each run used five Caliper workers, 20 prepared users per worker, the same fixed-rate controller, and a clean-state Fabric/IPFS restart. User registration was completed before the measured round. Figure 5 overlays every independent run for Caliper completion throughput and mean ledger latency; the summary markers and error bars indicate mean  $\pm$  SD. The comparison is confined to Fabric transaction-path payload cost rather than end-to-end storage-workflow latency.

**Table 7.** Ledger-path payload-placement results under matched 5- and 20-TPS conditions.

| Condition             | Ledger Condition     | Load/Payload                                                         | Terminal Per Run (Success/Failure) | Observed Send/Caliper Completion Rate (TPS) | Avg/Max Latency (s)                   |
|-----------------------|----------------------|----------------------------------------------------------------------|------------------------------------|---------------------------------------------|---------------------------------------|
| Low-load baseline     | Direct large-payload | n = 5; 300 tx; 5 TPS; 384,000 decoded bytes (512,000 Base64 chars)   | 300/0                              | 5.10/4.96 $\pm$ 0.09                        | 0.50 $\pm$ 0.01/<br>1.89 $\pm$ 0.49   |
| Low-load baseline     | Compact anchoring    | n = 5; 300 tx; 5 TPS; SHA-256 + real CID; IPFS prepared before round | 300/0                              | 5.10/5.10 $\pm$ 0.00                        | 0.60 $\pm$ 0.01/<br>1.15 $\pm$ 0.04   |
| 20-TPS capacity point | Direct large-payload | n = 3; 60 s; 20 TPS; 384,000 decoded bytes (512,000 Base64 chars)    | 1205/0                             | 20.10/14.20 $\pm$ 0.40                      | 31.62 $\pm$ 3.35/<br>43.80 $\pm$ 3.63 |
| 20-TPS capacity point | Compact anchoring    | n = 3; 60 s; 20 TPS; SHA-256 + real CID; IPFS prepared before round  | 1205/0                             | 20.10/19.40 $\pm$ 0.00                      | 0.22 $\pm$ 0.00/<br>2.05 $\pm$ 0.00   |

The direct condition carried a deterministic Base64 media payload of 384,000 decoded bytes, producing a 512,000-character field and a fixed SHA-256 value. The compact-anchor condition carried the same SHA-256 value and a real IPFS CID, while the media body remained outside the Fabric transaction. IPFS upload and CID retrieval verification were completed during run preparation and were excluded from Caliper latency. Both ledger conditions completed 300/300 transactions in every 5-TPS run. At 20 TPS, direct submission accumulated severe ledger delay and lower completion throughput, whereas compact anchoring remained close to the observed send rate with low ledger latency.

The result isolates the ledger effect of payload placement: Fabric stores compact event semantics, the SHA-256 anchor, and the CID, while IPFS carries the media object. At 20 TPS, all 1205 submitted transactions reached successful terminal outcomes in every run; the corresponding effective completion times were 84.9  $\pm$  2.4 s for the direct large-payload condition and 62.1  $\pm$  0.0 s for the compact-anchor condition.



**Figure 5.** Run-level ledger throughput and latency for direct large-payload and compact-anchor conditions. Individual points represent independent runs; markers and error bars indicate mean  $\pm$  SD ( $n = 5$  at 5 TPS;  $n = 3$  at 20 TPS).

#### 4.5. API-Boundary RabbitMQ Publication and Consumer-Enabled Completion

As shown in Table 8 and Figure 6, three Tencent PTS spike runs per mode used 1000 virtual users over 60 s. The synchronous endpoint executed the provenance request path, whereas the RabbitMQ endpoint published a persistent message to the durable queue and returned HTTP 202. Table 8 and Figure 6 therefore compare client-visible processing at two API boundaries. Figure 6 overlays the three PTS observations for each mode; the summary markers and error bars indicate mean  $\pm$  SD.

**Table 8.** Tencent PTS closed-loop results for synchronous and RabbitMQ publication modes.

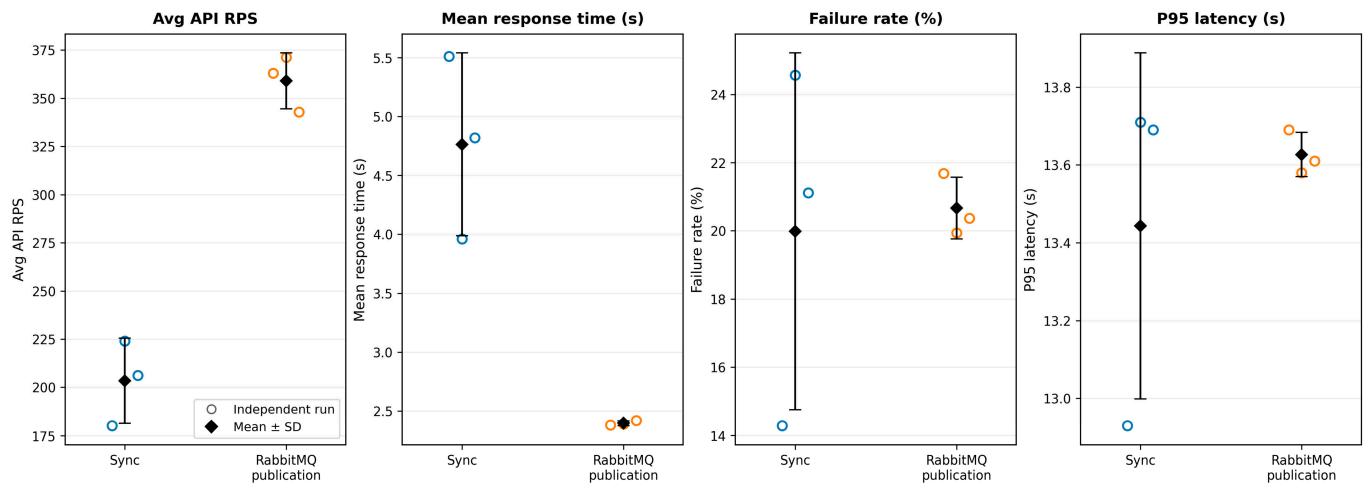
| Mode                 | Runs    | Total Requests (Closed-Loop) | Avg API RPS (Closed-Loop) | Peak API RPS      | Avg. Response Time (s) | Failure Rate (%) | P95 Latency (s)  |
|----------------------|---------|------------------------------|---------------------------|-------------------|------------------------|------------------|------------------|
| Sync                 | $n = 3$ | $13,095 \pm 1757$            | $203.41 \pm 22.07$        | $266.99 \pm 6.79$ | $4.76 \pm 0.78$        | $19.99 \pm 5.24$ | $13.44 \pm 0.44$ |
| RabbitMQ publication | $n = 3$ | $24,120 \pm 256$             | $359.00 \pm 14.56$        | $412.44 \pm 8.79$ | $2.40 \pm 0.02$        | $20.66 \pm 0.91$ | $13.63 \pm 0.06$ |

The RabbitMQ producer path recorded more client loops, sustained higher closed-loop API RPS, and reduced mean response time. Residual failure rate and P95 latency remained similar between modes. The measured effect is the earlier completion of the HTTP-to-queue publication path under the closed-loop workload. Accordingly, the PTS comparison supports a client-facing latency and request-turnover effect rather than an API-level reliability improvement.

JMeter and Tencent PTS served distinct roles. JMeter validated request composition, fixed-payload replay, and synchronized 1000-request emission from a Windows host. Tencent PTS provided the repeated cloud-side closed-loop comparison reported in Table 8.

API failures are generator-observed HTTP outcomes. In the RabbitMQ mode, successful requests correspond to persistent message publication and HTTP 202 responses. Queue

observations recorded ready-message accumulation and zero active consumers during the spike window.



**Figure 6.** Tencent PTS closed-loop ingress comparison under a 1000-VU burst. Individual points represent independent runs; markers and error bars indicate mean  $\pm$  SD ( $n = 3$  per mode).

The PTS runs characterize producer-side publication behavior for the evaluated single-replica gateway and RabbitMQ configuration. The consumer-enabled results reported in Section 4.5.3 complement this boundary by measuring queue drain, IPFS retention, Fabric commit, retry and failure outcomes, and total completion.

#### 4.5.1. Producer-Side Publication Scope

The PTS and JMeter measurements cover request receipt, message publication, publish-call completion, and HTTP response. The synchronous measurements cover request receipt and the configured provenance-service call. Queue depth and ready-message counts describe accumulated work at the broker.

The consumer-enabled protocol recorded message delivery, IPFS retention, Fabric commit, query visibility, and manual acknowledgement. Per-message timestamps across these transitions provide the admission-to-completion measurement sequence used in Section 4.5.3.

#### 4.5.2. Repeated JMeter Synchronized Spike Checks

Three 1000-user JMeter synchronized-spike checks were executed per mode at the same/uplink API. These runs used the fixed Base64 placeholder request body, 1000 users, a 0 s ramp-up, one loop, and a Synchronizing Timer grouping all users with a 30,000 ms timeout. They exercised the same API path under a reproducible local generator, with mean  $\pm$  standard deviation reported for  $n = 3$  runs per mode.

As summarized in Table 9, both modes completed all 1000 requests in each repeated JMeter run with zero aggregate JMeter-reported errors. The RabbitMQ-based asynchronous path reduced mean latency from  $2.286 \pm 0.246$  s to  $1.446 \pm 0.219$  s and reduced P95/P99 tail latency from  $6.624 \pm 3.993$  s/ $6.842 \pm 3.816$  s to  $2.990 \pm 0.380$  s/ $3.871 \pm 1.116$  s. Request-rate values varied substantially across the three runs; the JMeter comparison therefore emphasizes fixed-request completion and latency.

The JMeter checks provide fixed-request completion and latency evidence under a fixed-payload local replay. PTS and JMeter are reported separately because their load generators and traffic-delivery mechanisms differ. The asynchronous JMeter measurements characterize persistent broker publication and HTTP 202 response time.

**Table 9.** Repeated-run JMeter results under synchronized 1000-user spikes.

| Mode                        | Runs  | Throughput (req/s) | Avg Latency (s) | P95/P99 Latency (s)         | Error Rate (%) |
|-----------------------------|-------|--------------------|-----------------|-----------------------------|----------------|
| Synchronous                 | n = 3 | 90.56 ± 60.96      | 2.286 ± 0.246   | 6.624 ± 3.993/6.842 ± 3.816 | 0.00 ± 0.00    |
| RabbitMQ-based asynchronous | n = 3 | 124.10 ± 100.40    | 1.446 ± 0.219   | 2.990 ± 0.380/3.871 ± 1.116 | 0.00 ± 0.00    |

#### 4.5.3. Consumer-Enabled Queue-Drain and Completion Results

As summarized in Table 10, three independent 30-message runs followed the protocol in Section 3.5.2. All 90 messages were accepted and completed successfully, with no retry publications, producer failures, dead-letter outcomes, or missing IPFS/Fabric stage timestamps. Queue-drain times were 61.090, 61.808, and 62.504 s, yielding  $61.80 \pm 0.71$  s. Total completion times were 62.813, 63.077, and 63.069 s, yielding  $62.99 \pm 0.15$  s. Mean backend IPFS stage time across runs was  $0.0150 \pm 0.0006$  s, whereas mean Fabric commit time was  $2.0390 \pm 0.0006$  s.

**Table 10.** Consumer-enabled RabbitMQ completion results.

| Run       | Accepted/Completed | Queue-Drain Time (s) | Backend IPFS Stage Mean (s) | Fabric Commit Stage Mean (s) | Retries/Producer Failures/DLQ | Total Completion Time (s) |
|-----------|--------------------|----------------------|-----------------------------|------------------------------|-------------------------------|---------------------------|
| 1         | 30/30              | 61.090               | 0.014951                    | 2.039415                     | 0/0/0                         | 62.813                    |
| 2         | 30/30              | 61.808               | 0.015643                    | 2.038349                     | 0/0/0                         | 63.077                    |
| 3         | 30/30              | 62.504               | 0.014496                    | 2.039372                     | 0/0/0                         | 63.069                    |
| Mean ± SD | 90/90 total        | 61.801 ± 0.707       | 0.015030 ± 0.000578         | 2.039045 ± 0.000603          | 0/0/0                         | 62.986 ± 0.150            |

The completion results separate message admission from downstream processing. HTTP 202 confirms persistent queue publication, whereas completion additionally requires queue delivery, IPFS retention, ledger confirmation, query visibility, and acknowledgement. Under one consumer with prefetch 1, the Fabric commit stage was approximately two orders of magnitude longer than the backend IPFS stage and accounted for most of the measured per-message processing interval. The reported quantity is serial completion timing under one consumer; multi-consumer capacity remains a distinct scaling variable.

#### 4.5.4. Summary-Level Statistical Checks

Table 11 reports effect estimates from the repeated-run evidence. Ledger-path payload-placement results are summarized by observed differences because the 20-TPS compact-anchor runs produced zero failures and no variation at the reported precision in completion throughput or mean latency. Welch intervals and *p* values are reported for the PTS contrasts. The consumer-enabled experiment is summarized descriptively because it evaluates completion and stage timing under a fixed one-consumer protocol. Differences are calculated as the proposed path minus the baseline path; negative latency values indicate lower delay.

Failure outcomes are defined at their measurement boundaries. Caliper failures are terminal ledger-client outcomes associated with gateway waiting, endorsement or commit delay, validation failure, or client backlog under saturation. PTS and JMeter failures are HTTP outcomes observed by the load generator. Consumer-enabled outcomes distinguish producer failure, retry publication, dead-letter routing, missing-stage timestamps, and failure to reach query-visible acknowledgement. Component-level attribution requires synchronized gateway, broker, object-store, and Fabric-event traces.

**Table 11.** Summary-level checks for ledger-path, ingress, and consumer-completion results.

| Comparison                                      | Difference/Observation                                                                                                                              | Note                                        | Interpretation                                                                           |
|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|------------------------------------------------------------------------------------------|
| Ledger-path completion (Table 7)                | Both ledger conditions completed all terminal transactions at 5 TPS; zero terminal failures also occurred in the 20-TPS runs.                       | Descriptive                                 | Fixed denominators and terminal outcomes are reported explicitly.                        |
| 20-TPS ledger-path throughput (Table 7)         | Compact-anchor minus direct large-payload: +5.20 TPS.                                                                                               | n = 3 per condition                         | Compact anchoring retained higher ledger-path completion throughput.                     |
| 20-TPS mean ledger latency (Table 7)            | Compact-anchor minus direct large-payload: −31.40 s.                                                                                                | n = 3 per condition                         | Carrying the media field through Fabric produced severe ledger-path delay.               |
| Closed-loop API RPS (Table 8)                   | Async minus sync: +155.59 RPS [110.49, 200.69].                                                                                                     | Welch $p = 0.0011$                          | Higher HTTP-to-queue processing under the closed-loop load model.                        |
| Mean response time (Table 8)                    | Async minus sync: −2.36 s [−4.30, −0.42].                                                                                                           | Welch $p = 0.0345$                          | Lower client-visible delay.                                                              |
| Residual failure rate and P95 latency (Table 8) | +0.67 pp failure; +0.19 s P95.                                                                                                                      | Welch $p = 0.846/0.533$                     | Residual failure rate and P95 latency remained statistically similar.                    |
| Consumer-enabled completion                     | 90/90 messages completed; 0 retry publications; 0 producer failures; 0 dead-letter outcomes.                                                        | n = 3 runs; 30 messages per run             | Confirms downstream completion under the fixed one-consumer protocol.                    |
| Consumer-enabled stage timing                   | Queue drain $61.80 \pm 0.71$ s; backend IPFS stage $0.0150 \pm 0.0006$ s; Fabric commit $2.0390 \pm 0.0006$ s; total completion $62.99 \pm 0.15$ s. | Descriptive mean $\pm$ SD across three runs | Distinguishes client-facing admission from downstream retention and ledger confirmation. |

#### 4.6. Interim Summary of Findings

The results answer the three research questions for the fixed prototype. For RQ1, compact ledger anchoring and direct large-payload submission both completed reliably at 5 TPS, while compact anchoring preserved lower ledger latency and higher completion throughput at 20 TPS. For RQ2, the 30- and 50-TPS rounds formed the stable region, 100 TPS marked the transition point, and higher loads became delay-dominated. For RQ3, persistent RabbitMQ publication increased closed-loop API processing and reduced mean response time, while residual failure rate and P95 latency remained statistically similar between modes. In the consumer-enabled experiment, all 90 messages reached IPFS retention, successful Fabric commit, query visibility, and manual acknowledgement without retry publication, producer failure, or dead-letter outcome; mean queue-drain and total completion times were  $61.80 \pm 0.71$  s and  $62.99 \pm 0.15$  s.

Together, the experiments quantify Fabric write behavior, HTTP-to-queue admission, and downstream completion as separate stages of the media-evidence pipeline.

## 5. Discussion

The ledger and ingress experiments expose two different system costs. Carrying media bytes through Fabric propagates object size into endorsement, ordering, validation, and state persistence, whereas compact anchoring confines ledger work to bounded semantics, a SHA-256 value, and a CID. Producer-side persistent publication shortens the client request path and exposes published work as queue backlog, while consumer-enabled measurements quantify queue drain and downstream completion.

In the prototype, IPFS functions as an institution-managed content-addressed retention layer. It provides CID-based binding between an evidence object and the Fabric

record. The ledger-path payload-placement experiment isolates Fabric transaction-path cost because IPFS upload and retrieval verification occur before the measured Caliper round. S3-compatible object storage or MinIO provides an alternative retention tier; the architectural requirement includes retrievable evidence bytes and a verifiable anchor. End-to-end retention latency additionally includes object upload, retrieval, and availability management.

The producer-only RabbitMQ path recorded more closed-loop requests and lower mean API response time, while residual failure rate and P95 latency remained close to the synchronous path. In the consumer-enabled experiment, all 90 messages completed IPFS retention, successful Fabric commit, query visibility, and manual acknowledgement without retry, producer failure, or dead-letter outcome. The mean queue-drain time of  $61.80 \pm 0.71$  s and total completion time of  $62.99 \pm 0.15$  s show that client acknowledgement preceded downstream completion. The Fabric commit stage ( $2.0390 \pm 0.0006$  s) was substantially longer than the backend IPFS stage ( $0.0150 \pm 0.0006$  s) and dominated the measured per-message processing interval.

For post-exam review, appeal handling, and institutional audit, the system state should distinguish submitted, queue-published, object-retained, ledger-committed, query-visible, and acknowledged evidence. The experiments measure producer-side queue publication, controlled consumer completion, and Fabric ledger behavior as separate stages.

The write-path results define a topology-specific capacity envelope. The 30- and 50-TPS rounds combined high ledger success with sub-second mean latency, while the 100-TPS round retained low latency but showed a measurable decline in success. Higher fixed-rate rounds became delay dominated and accumulated failures. Capacity planning for integrity-sensitive evidence services therefore combines successful write throughput, ledger success rate, latency, and backlog.

The broader implication is a workload-sensitive provenance principle. Final credentials are compact outcome artifacts, whereas suspicious-event evidence is process-level, media-rich, privacy-sensitive, and review-driven. Aligning storage placement, verification semantics, and admission behavior with this workload shape avoids forcing heterogeneous educational records through a single persistence and service model.

### *5.1. Cloud Deployment Implications*

Three deployment principles follow from the measured paths. First, Fabric should retain bounded semantics and verification references, while the object layer absorbs media size. Second, queue publication and ledger completion require separate status transitions and metrics. Third, consumer concurrency should be aligned with the measured provenance write capacity and observed queue-drain behavior.

Direct media deposition makes endorsement, ordering, validation, and state persistence scale with the media payload. Compact anchoring shifts media capacity to the object-retention tier and keeps ledger growth tied to bounded anchors. RabbitMQ adds a persistent-message admission buffer; in the consumer-enabled experiment, Fabric confirmation accounted for substantially more downstream time than the backend IPFS stage, while consumer, retry, and dead-letter resources determine scale-out behavior.

These implications can extend to other institutional services that handle review-oriented media evidence when the workload preserves the same characteristics: large objects, compact integrity anchors, delayed review, and burst-prone arrival. Real-time control workloads and different trust boundaries form separate deployment classes with stricter timing and governance requirements.

### 5.2. Multi-Host Deployment and Scale-Out Considerations

Although the Fabric configuration exposes two peer endpoints and one orderer, all three share one provenance host. A multi-host consortium changes endorsement fan-out, ordering traffic, TLS overhead, replicated state, state-database writes, and network latency. Block size and block timeout can also shift the balance between batching efficiency and confirmation delay. The 30–50 TPS stable region and the 100-TPS transition point provide baselines for this topology and a reference point for subsequent scale-out experiments.

The one-consumer experiment confirms functional downstream completion under the evaluated workload: persistent messages were retrieved, objects were retained, Fabric transactions were committed, records became query visible, and acknowledgements were issued without retry or dead-letter outcomes. Stable drain at a larger scale remains dependent on consumer count, prefetch, retry policy, broker policy, endorsement latency, and the provenance-write capacity of the deployment.

Scale-out evaluation should vary Fabric endorsement policies, peer and orderer placement, state-database deployment, object size, consumer count, prefetch, retry policy, and broker topology while recording CID retrieval results, queue-depth trajectories, dead-letter outcomes, drain time, and gateway connection pressure.

### 5.3. Storage Resilience, Privacy, and Million-Scale Retention

A million-scale evidence repository shifts the scaling problem from single-event anchoring to indexing, retention, and audit management. The ledger record should remain compact, while exam-level, course-level, or term-level indexes should support retrieval without requiring full-ledger scans. Cold evidence can be moved to lower-cost encrypted retention tiers, while high-risk or appeal-relevant evidence remains pinned and periodically checked. Retention policy should define when objects are preserved, archived, anonymized, or deleted from managed off-chain storage while the chain retains only the minimum necessary provenance semantics.

Sensitive evidence requires privacy protection beyond CID binding. Screen-capture evidence and detector artifacts should be encrypted before off-chain retention, access should be mediated by institutional authorization instead of CID possession, and every retrieval, verification, export, or deletion request should be logged. Key-management metadata, such as key version and retention period, should be bound to the evidence record without exposing raw personal data on-chain. Where identifiers are not required for public audit, hashed or pseudonymized identifiers should be used to reduce disclosure risk.

Decentralized identifiers and verifiable credentials can strengthen the identity and authorization boundary without changing the Fabric/IPFS retention path. Blockchain-based verifiable decentralized identity has been used to bind roles and authorities in a distributed industrial setting [25]. In a cross-institutional examination setting, the same design principle could bind candidates, invigilators, reviewers, and institutional authorities to verifiable roles, while Fabric records preserve evidence actions and IPFS retains the media object. Such an extension should avoid storing personal attributes on-chain; the ledger should record identity proofs, role assertions, key versions, and audit actions rather than raw identity documents.

At deployment scale, availability and privacy depend on multi-host Fabric governance, replicated or backed-up object retention, retrieval audits, active consumer telemetry, dead-letter reconciliation, key rotation, and controlled evidence release. These controls extend the architecture without placing media bodies on the ledger.

#### 5.4. External Validity and Prototype Boundary

The fixed topology strengthens internal comparison by holding the provenance substrate and service resources constant. Fabric, IPFS, and Caliper ran on one 4-vCPU/8-GB CVM, while the API and RabbitMQ tier ran in TKE. The provenance host's 5 Mbps public-bandwidth allocation applied to its public interface, while the segmented benchmark paths followed the routes reported in Section 3.6. Multi-host endorsement, orderer scale-out, state-database placement, cross-region retention, and dedicated benchmark-client separation can shift throughput and latency boundaries.

The repeated runs use controlled payloads and a limited set of traffic shapes. The direct Caliper condition carries a deterministic media-sized Base64 field; the compact-anchor condition measures ledger anchoring after the IPFS object and CID have been prepared; the RabbitMQ spike runs measure producer-only queue publication; and the consumer-enabled runs use one manual-acknowledgement consumer, prefetch 1, and a newly registered user per run. External validation should add empirical object-size distributions, multi-consumer queue-drain traces, fault-injected retry and dead-letter outcomes, and synchronized component-level failure logs. Table 12 consolidates the security and privacy boundary of the evaluated prototype and the corresponding deployment-scale extensions.

**Table 12.** Security and privacy boundary of the evaluated prototype.

| Concern                                   | Evaluated Prototype Boundary                                                                                                                                                                                                                                                                                  | Deployment-Scale Extension                                                                                                                                                           |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Metadata tampering                        | Fabric endorsement and key-history queries make committed metadata changes traceable within the permissioned channel.                                                                                                                                                                                         | Multi-organization adversarial governance and endorsement-policy sweeps.                                                                                                             |
| Candidate or retained-byte substitution   | SHA-256 comparison verifies a supplied candidate and the exact bytes retrieved from the recorded CID.                                                                                                                                                                                                         | Periodic large-scale CID retrieval audits and signed audit reports.                                                                                                                  |
| Off-chain object privacy and CID exposure | Institution-managed IPFS retention and gateway-mediated access keep identifiers and objects within the controlled prototype.                                                                                                                                                                                  | Encryption benchmarking, CID-leakage testing, and access-token abuse evaluation.                                                                                                     |
| Authentication and key handling           | Application endpoints operated within the institution-controlled prototype environment; SHA-256 verification uses no secret key.                                                                                                                                                                              | mTLS or signed requests, HSM/KMS-backed signing keys, rotation, and compromise simulation.                                                                                           |
| Replay and idempotency                    | Evidence identifiers, timestamps, and hash/CID anchors support duplicate analysis and temporal audit.                                                                                                                                                                                                         | Active replay testing and idempotency-key stress tests.                                                                                                                              |
| Queue failure                             | Producer-side spike runs used one durable queue, persistent messages, a producer-only gateway, and zero consumers. The completion runs used one manual-acknowledgement consumer, prefetch 1, a maximum of three retries, and a dedicated DLQ; all 90 messages completed without retry or dead-letter outcome. | Multi-consumer fault injection, retry exhaustion, DLQ reconciliation, idempotency validation, and queue-drain and completion timing under broker or provenance-service interruption. |
| Audit-log abuse                           | Query, history, and verification presentation are mediated by the institutional service boundary.                                                                                                                                                                                                             | Multi-party approval and insider-misuse evaluation.                                                                                                                                  |

#### 5.5. Future Research Directions

Four extensions follow directly from the current boundary. First, DID- and verifiable credential-based authorization can bind candidates, invigilators, reviewers, and institutional authorities across examination providers while preserving selective disclosure [25]. Second, provenance verification and query optimization should be evaluated for million-

scale evidence metadata, including verifiable query results and indexing structures for exam-, course-, time-, and case-level retrieval [21,22].

Third, privacy-preserving evidence verification should be strengthened through encrypted object retention, controlled disclosure, and periodic CID-level byte audits. Fourth, deployment-scale validation should extend the measured single-consumer experiment to multi-consumer scaling, varied prefetch, fault-injected retry exhaustion and dead-letter reconciliation, multi-host Fabric, replicated IPFS or object-store baselines, and per-message admission-to-completion telemetry.

## 6. Conclusions

This article presented a Kubernetes-deployed media-evidence provenance pipeline combining compact Fabric anchoring, IPFS object retention, SHA-256 evidence verification, producer-side RabbitMQ publication, and consumer-enabled downstream completion. The evaluation separated ledger payload placement, provenance-write saturation, API-boundary publication, and consumer-enabled queue drain and downstream completion under a fixed single-host provenance substrate and a separate Kubernetes service tier.

For RQ1, the matched 5-TPS baseline showed that both ledger conditions completed the same 300-transaction workload. At the 20-TPS capacity-separation point, compact anchoring of a pre-retained IPFS object maintained  $19.40 \pm 0.00$  TPS with  $0.22 \pm 0.00$  s mean ledger latency, whereas direct large-payload on-chain submission achieved  $14.20 \pm 0.40$  TPS with  $31.62 \pm 3.35$  s mean latency under the same observed send rate. The result supports excluding media bytes from the Fabric transaction path while retaining the SHA-256 anchor and CID on-chain. End-to-end retention performance remains a separate system-level measure because IPFS preparation preceded the measured compact-anchor round.

For RQ2, the 30- and 50-TPS rounds formed the stable region, 100 TPS marked the transition point, and higher loads were dominated by latency and failure accumulation. For RQ3, the producer-only RabbitMQ path recorded more closed-loop client requests and reduced mean API response time, while residual failure rate and P95 latency remained close between modes. The consumer-enabled experiment completed all 90 admitted messages through IPFS retention, successful Fabric commit, query visibility, and manual acknowledgement, with no retries, producer failures, or dead-letter outcomes. Mean queue-drain and total completion times were  $61.80 \pm 0.71$  s and  $62.99 \pm 0.15$  s; the Fabric commit stage dominated the measured downstream interval. The results show client admission, object retention, ledger commit, query visibility, acknowledgement, and queue drain as distinct operational states.

**Supplementary Materials:** The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/electronics15153478/s1>. Supporting code, configurations, benchmark data, and figure sources accompany this article as Supplementary Materials.

**Author Contributions:** Conceptualization, H.W.; methodology, H.W.; software, H.W.; validation, H.W.; formal analysis, H.W.; investigation, H.W.; data curation, H.W. and L.W.; visualization, H.W.; writing—original draft preparation, H.W.; writing—review and editing, Z.S. and K.A.Z.A.; supervision, Z.S. and K.A.Z.A.; resources, Z.S. and K.A.Z.A.; project administration, Z.S.; funding acquisition, H.W. and L.W. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the Educational Science Planning Project of Guangdong Province (Higher Education Special Project), grant number 2024GXJK679; the 2025 Annual Project of the Industry-Education Integration and School-Enterprise Cooperation Research Branch of the Guangdong Society of Higher Education, grant number 25-GDGJXQ-07; the 2025 Guangdong Provincial Undergraduate Teaching Quality and Teaching Reform Project, project number 2025GDJG05, approved under document Yue Jiao Gao Han [2026] No. 4; and the 2024 Guangdong Provincial

Undergraduate Teaching Quality and Teaching Reform Project—Special Talent Development Program (Outstanding HarmonyOS Application-Oriented Talent Cultivation Program), approved under document Yue Jiao Gao Han [2024] No. 30.

**Data Availability Statement:** Data supporting the reported results are contained within the article and its Supplementary Materials.

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

## References

- Noorbehbahani, F.; Mohammadi, A.; Aminazadeh, M. A systematic review of research on cheating in online exams from 2010 to 2021. *Educ. Inf. Technol.* **2022**, *27*, 8413–8460. [\[CrossRef\]](#) [\[PubMed\]](#)
- Kaddoura, S.; Gumaei, A. Towards effective and efficient online exam systems using deep learning-based cheating detection approach. *Intell. Syst. Appl.* **2022**, *16*, 200153. [\[CrossRef\]](#)
- Yaqub, W.; Mohanty, M.; Suleiman, B. Privacy-preserving online proctoring using image-hashing anomaly detection. In Proceedings of the International Wireless Communications and Mobile Computing Conference (IWCMC), Dubrovnik, Croatia, 30 May–3 June 2022; pp. 1113–1118. [\[CrossRef\]](#)
- Hylton, K.; Levy, Y.; Dringus, L.P. Utilizing webcam-based proctoring to deter misconduct in online exams. *Comput. Educ.* **2016**, *92–93*, 53–63. [\[CrossRef\]](#)
- Dendir, S.; Maxwell, R.S. Cheating in online courses: Evidence from online proctoring. *Comput. Hum. Behav. Rep.* **2020**, *2*, 100033. [\[CrossRef\]](#)
- Cardenas-Quispe, M.A.; Pacheco, A. Blockchain ensuring academic integrity with a degree verification prototype. *Sci. Rep.* **2025**, *15*, 9281. [\[CrossRef\]](#) [\[PubMed\]](#)
- Abdelsalam, M.; Shokry, M.; Idrees, A.M. A proposed model for improving the reliability of online exam results using blockchain. *IEEE Access* **2024**, *12*, 7719–7733. [\[CrossRef\]](#)
- Alammary, A.; Alhazmi, S.; Almasri, M.; Gillani, S. Blockchain-based applications in education: A systematic review. *Appl. Sci.* **2019**, *9*, 2400. [\[CrossRef\]](#)
- Trautwein, D.; Raman, A.; Tyson, G.; Castro, I.; Scott, W.; Schubotz, M.; Gipp, B.; Psaras, Y. Design and evaluation of IPFS: A storage layer for the decentralized web. In Proceedings of the ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, 22–26 August 2022; pp. 739–752. [\[CrossRef\]](#)
- Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger Fabric: A distributed operating system for permissioned blockchains. In Proceedings of the 13th EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15. [\[CrossRef\]](#)
- Thakkar, P.; Nathan, S.; Viswanathan, B. Performance benchmarking and optimizing Hyperledger Fabric blockchain platform. In Proceedings of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Milwaukee, WI, USA, 25–28 September 2018; pp. 264–276. [\[CrossRef\]](#)
- Zang, H.; Kim, H.; Kim, J. Blockchain-based decentralized storage design for data confidence over cloud-native edge infrastructure. *IEEE Access* **2024**, *12*, 50083–50099. [\[CrossRef\]](#)
- Rani, D.; Gill, N.S.; Gulia, P.; Yahya, M.; Ahanger, T.A.; Hassan, M.M.; Abdallah, F.B.; Shukla, P.K. A secure digital evidence preservation system for an IoT-enabled smart environment using IPFS, blockchain, and smart contracts. *Peer-to-Peer Netw. Appl.* **2025**, *18*, 5. [\[CrossRef\]](#)
- Liu, S.; Zheng, Q. A study of a blockchain-based judicial evidence preservation scheme. *Blockchain Res. Appl.* **2024**, *5*, 100192. [\[CrossRef\]](#)
- Nizamuddin, M.; Salah, K.; Azad, M.A.; Arshad, J.; Rehman, M.H. Decentralized document version control using Ethereum blockchain and IPFS. *Comput. Electr. Eng.* **2019**, *76*, 183–197. [\[CrossRef\]](#)
- Deng, S.; Zhao, H.; Huang, B.; Zhang, C.; Chen, F.; Deng, Y.; Yin, J.; Dustdar, S.; Zomaya, A.Y. Cloud-native computing: A survey from the perspective of services. *Proc. IEEE* **2024**, *112*, 12–46. [\[CrossRef\]](#)
- Cao, H.; Liu, X.; Guo, H.; He, J.; Liu, X. QueueFlower: Orchestrating microservice workflows via dynamic queue balancing. In Proceedings of the IEEE International Conference on Web Services (ICWS), Shenzhen, China, 7–13 July 2024; pp. 1293–1299. [\[CrossRef\]](#)
- Vaño, R.; Lacalle, I.; Sowiński, P.; S-Julián, R.; Palau, C.E. Cloud-native workload orchestration at the edge: A deployment review and future directions. *Sensors* **2023**, *23*, 2215. [\[CrossRef\]](#) [\[PubMed\]](#)
- Durner, D.; Leis, V.; Neumann, T. Exploiting cloud object storage for high-performance analytics. *Proc. VLDB Endow.* **2023**, *16*, 2769–2782. [\[CrossRef\]](#)
- Alharthi, S.; Alshamsi, A.; Alseiyari, A.; Alwarafy, A. Auto-scaling techniques in cloud computing: Issues and research directions. *Sensors* **2024**, *24*, 5551. [\[CrossRef\]](#) [\[PubMed\]](#)

21. Deng, J.; Peng, Z.; Zhang, C.; Gu, S.; Xie, X.; Xiao, B. vProChain: Efficient provenance verification in Industrial Internet of Things (IIoT). *IEEE Internet Things J.* **2026**, *13*, 34813–34829. [[CrossRef](#)]
22. Wu, H.; Tang, Y.; Shen, Z.; Tao, J.; Lin, C.; Peng, Z. TELEX: Two-level learned index for rich queries on enclave-based blockchain systems. *IEEE Trans. Knowl. Data Eng.* **2025**, *37*, 4299–4313. [[CrossRef](#)]
23. Górski, T. Software architecture description in original software publications. *Softw. Impacts* **2026**, *27*, 100802. [[CrossRef](#)]
24. Wang, H.; Shukur, Z.; Zainol Ariffin, K.A.; Xiao, R.; Wang, L. Online exam cheating detection and blockchain trusted deposit based on YOLOv12. *Sci. Rep.* **2025**, *15*, 33236. [[CrossRef](#)] [[PubMed](#)]
25. Xu, W.; Deng, J.; Yu, J.; Mao, S.; Li, Y.; Peng, Z.; Xiao, B. Blockchain-based verifiable decentralized identity for intelligent flexible manufacturing. *IEEE Internet Things J.* **2025**, *12*, 32366–32378. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.