Academy for Global Science and Innovation (AGSI)

ARTICLE

A Networked Consensus Protocol for Collaborative Model Governance with Auditable Lineage in Data Platforms

Pham Minh Khang[1] and Vo Thanh Dat[2]

[1]Saigon Institute of Digital Technology, Department of Computer Science, Le Duan Boulevard, Ho Chi Minh City 700000, Vietnam

[2]Hanoi College of Advanced Computing, Department of Software Engineering, Kim Ma Street, Ba Dinh District, Hanoi 100000, Vietnam

Abstract

Diffusion-based image editors have become central tools for interactive visual manipulation, combining probabilistic generative models with flexible conditioning mechanisms. Many practical editing scenarios require simultaneous control of global image style and preservation of local scene structure, for example when re-rendering a photograph in a target artistic style while maintaining object identities, spatial layout, and fine geometric details. However, the latent spaces employed by diffusion models and their associated encoders often entangle style-related and structure-related factors, and naive conditioning or prompt-based guidance tends to either over-write local content or fail to propagate a coherent global appearance. This work studies latent-space control in diffusion-based editors with a particular focus on the trade-off between global style changes and local structure preservation. The paper develops a geometric perspective on latent diffusion spaces, where style manipulations are modeled as movements along a low-dimensional subspace while local structure is treated as a set of constraints induced by the image decoder and the diffusion denoising dynamics. Building on this perspective, we describe optimization-based formulations that explicitly balance style and structural losses, and we interpret several existing heuristic editing strategies as approximations to these formulations. We then outline practical algorithms that combine subspace projections, attention-weighted locality masks, and time-dependent control signals injected along the diffusion trajectory. Experimental scenarios are discussed qualitatively to illustrate how different design choices affect the balance between global coherence and local fidelity, and limitations and possible extensions of the approach are examined in detail.

1. Introduction

Collaborative model governance refers to the coordinated processes by which organizations specify, enforce, and audit the policies that control how machine learning models are created, modified, evaluated, approved, deployed, monitored, and retired [1]. In contemporary data platforms, governance spans a heterogeneous set of services that include dataset registries, feature stores, notebook and pipeline orchestration engines, model registries, experiment tracking services, deployment controllers, and monitoring and incident response systems. Each service often maintains its own logs and metadata, which can be consistent within a boundary but inconsistent across boundaries. The governance problem becomes more complex when multiple stakeholders participate with different authority scopes and when actions must be demonstrably compliant with internal controls and external regulations. Governance is not solely a question of access control; it is a question of capturing the rationale and evidence behind decisions and of ensuring that the lineage linking a deployed model to the data and code that produced it is auditable and resistant to tampering.

Many existing approaches treat governance as a layer of approval workflows over a centralized database [2]. This works well when a single administrative domain controls the entire platform, when the threat model is mostly accidental error, and when auditors trust the administrators to curate records. However, collaborative model development frequently spans organizational boundaries and supply chains. Data may be provided by one party, feature transformations by another, model architecture by a third, and deployment by a fourth. Even within a single organization, large platforms often operate as federations of teams with partially independent control planes. A central governance database becomes a single point of trust and a single point of failure, both operationally and for audit. In such settings, a networked approach that provides shared consistency guarantees, explicit evidence binding, and decentralized verification can reduce reliance on unilateral administrative assertions [3].

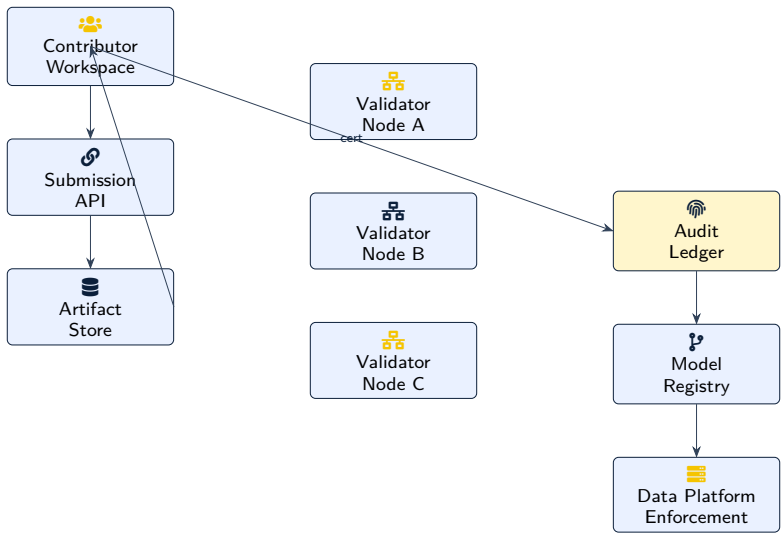


Figure 1. High-level architecture of a networked consensus layer coordinating collaborative model governance: submitted artifacts are certified by a validator overlay, committed to an audit ledger, and propagated to the model registry for platform enforcement.

Table 1. Notation used in the proposed consensus protocol and lineage model.

Symbol	Description	Type	Example Value
G	Governance network graph	Set of nodes and edges	64 validator nodes
M	Model artifact under governance	Versioned object	Model v3.1.4
$L(M)$	Lineage record associated with a model	Ordered event log	152 events
C	Consensus committee for a proposal	Subset of nodes	11 voting members
τ	Block time for consensus round	Time interval	2 seconds
f	Maximum tolerated faulty nodes	Scalar	$f = \lfloor (n-1)/3 \rfloor$

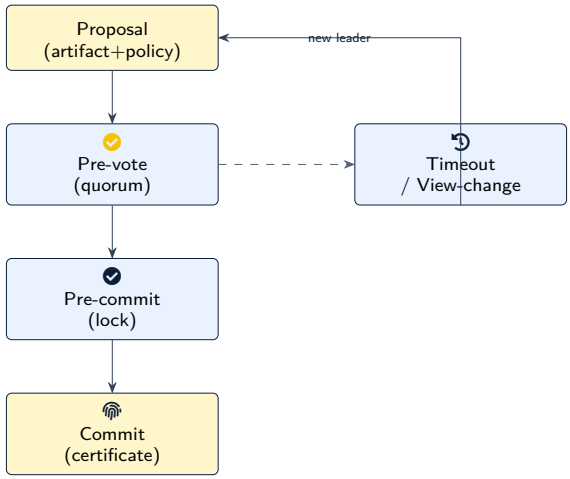


Figure 2. Compact consensus round for governance decisions: a proposal moves through pre-vote and pre-commit to produce a commit certificate; timeout triggers a view-change to maintain liveness under partial synchrony.

A second difficulty is that governance decisions depend on a rich set of artifacts that are not naturally captured by relational metadata alone. A training run may depend on a dataset snapshot, a set of feature definitions, a container image, a pipeline graph, hyperparameters, and a policy bundle. The run may produce evaluation results, interpretability reports, fairness metrics, robustness tests, and security scans. Governance requires that these artifacts be referenced in a stable way, that their integrity be verifiable, and that the relationships among them be reconstructible. Lineage graphs address the relationship modeling problem, while cryptographic content addressing addresses integrity. Yet lineage alone does not provide agreement on which governance actions are valid, nor does it provide a deterministic policy state that different parties can rely on [4]. Conversely, a consensus log without a lineage substrate risks becoming an opaque chain of approvals that cannot be independently substantiated.

This paper proposes a protocol that treats collaborative model governance as a replicated state machine whose input is a stream of governance transactions and whose state includes policy definitions, role and authority assignments, model and dataset registries, and references to evidence bundles stored in a content-addressed lineage substrate. The protocols key design choice is to couple governance consensus with lineage anchoring: each governance transaction commits to a set of lineage nodes and their hashes, and the consensus output commits to both the governance decision and the precise artifact graph that substantiates it. This yields an auditable record that can be verified by external auditors or internal compliance teams without requiring privileged access to mutable databases. The protocol is designed for data platforms where the execution of pipelines and storage of artifacts occur off-chain, while the governance log maintains the authoritative commitments and admissibility checks.

The paper focuses on a practical threat model in which some participants may behave arbitrarily, including by attempting to approve models without required evidence, by modifying lineage metadata after the fact, or by censoring governance actions [5]. It also considers benign but common failure modes such as network partitions, retries that create duplicate submissions, clock skew, and inconsistent identity mapping across services. The protocol aims to provide safety, meaning that no two honest participants finalize conflicting governance states,

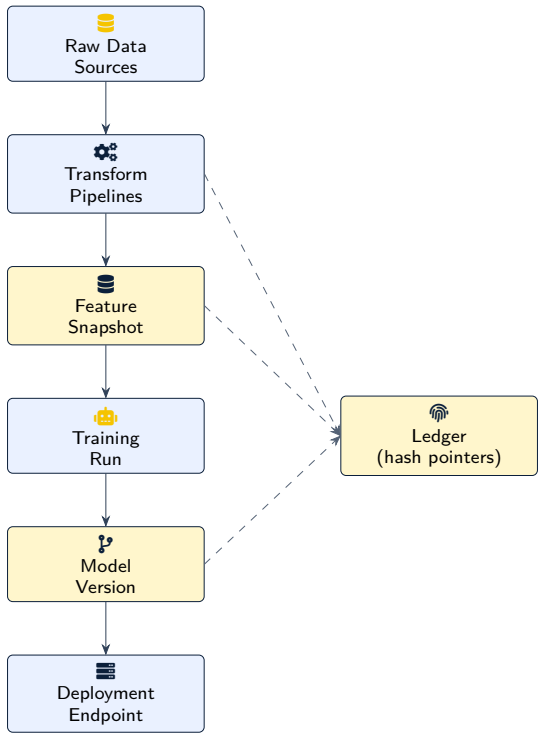


Figure 3. Auditable lineage as a content-addressed DAG: each transformation, feature snapshot, and model version emits a hash pointer committed to an immutable ledger, enabling end-to-end provenance for deployments.

and audit completeness, meaning that finalized decisions are bound to verifiable evidence and to a lineage graph that can be reconstructed. Liveness is considered under partial synchrony assumptions typical of Byzantine fault tolerant systems.

The remainder of the paper is organized into sections that formalize the system and threat models, define governance and lineage objects, describe the consensus protocol and transaction semantics, analyze correctness and performance, and discuss implementation and evaluation in real data platforms. The goal is to articulate a coherent design space and a concrete protocol blueprint rather than to argue that one specific implementation dominates all alternatives. The protocols contributions are framed as a set of mechanisms that can be adopted selectively, with trade-offs made explicit in terms of fault tolerance, operational overhead, and verification cost [6].

2. System Model and Threat Model

A data platform in scope is assumed to provide persistent storage for datasets and model artifacts, compute services for transformation and training, metadata services for cataloging, and control-plane services for deployment. Let the platform be partitioned into governance domains, where each domain corresponds to an administrative boundary with its own identity provider, key management practices, and operational policies. Domains interact through a governance network in which selected nodes participate in consensus. The governance network is responsible for ordering governance transactions and deriving a canonical governance state.

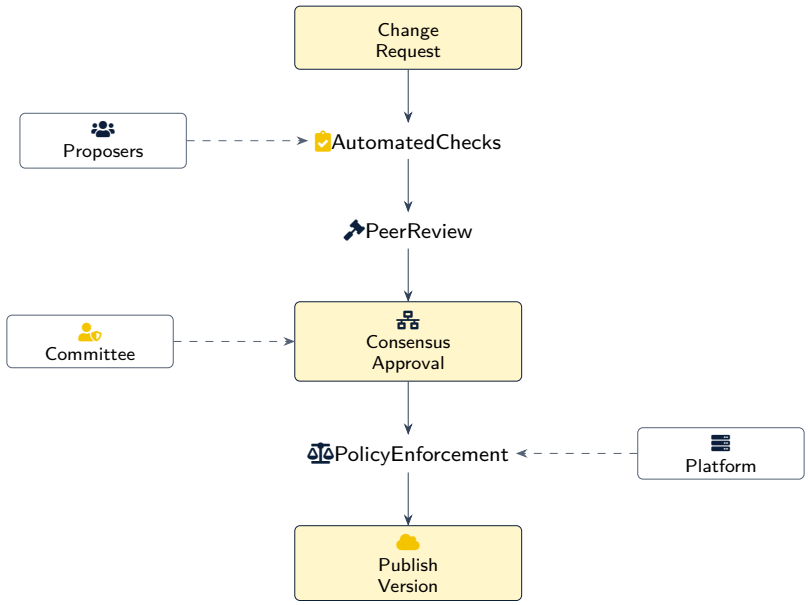


Figure 4. Collaborative governance workflow from change request to publication: automated checks and peer review feed a consensus approval that drives policy enforcement and controlled version release.

Artifacts and large evidence bundles are stored in external object stores, artifact registries, or repositories, but are referenced by cryptographic digests. The protocol is designed to function across heterogeneous infrastructure, including hybrid cloud and on-prem environments, with varying degrees of connectivity [7].

We define a set of principals. Human principals include model developers, reviewers, risk owners, and operators. Service principals include pipeline executors, scanning services, evaluation harnesses, and deployment controllers. Each principal is associated with a public key and an identity assertion that can be validated by one or more trust anchors. In federated settings, identity is mapped across domains using signed assertions that bind local identifiers to global principal identifiers. The protocol does not assume a single identity provider; it assumes that identities are represented as verifiable credentials or signed attestations that can be checked during transaction validation [8]. Key rotation is modeled as an update to an identity binding, which itself is a governance action recorded in the consensus log.

The state machine replicated by the governance network maintains several classes of objects. Policy objects define admissibility predicates, such as required review roles, required evaluation metrics thresholds, required security scans, and required data usage constraints. Authority objects define which principals can propose, endorse, or finalize particular actions, potentially with multi-signature requirements. Registry objects represent logical entities such as datasets, feature definitions, model versions, deployment targets, and monitoring configurations. Evidence objects represent signed statements about artifacts, such as a scanner attesting that a container image has no critical vulnerabilities, or an evaluation harness attesting to measured performance on a held-out dataset snapshot [9]. Lineage objects represent nodes in a provenance graph that relate inputs, transformations, and outputs.

We assume that the governance network consists of n consensus nodes, where up to f may

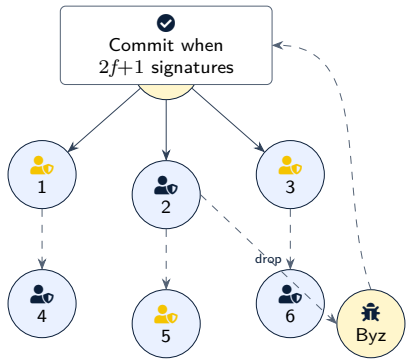


Figure 5. Validator quorum sketch: a proposer disseminates a proposal to nearby peers; commit requires a threshold of signatures while a faulty participant is isolated from the certificate formation path.

Table 2. Roles participating in collaborative model governance across the network.

Role	Primary Responsibilities	Permissions	Typical Actor
Model Owner	Define objectives, approve deployments	Propose and vote on changes	Product team lead
Data Steward	Validate data quality and policies	Attach data lineage evidence	Data governance team
Auditor	Inspect history and controls	Read-only access to all logs	Internal compliance
Platform Operator	Maintain infrastructure	Manage node configuration	MLOps engineer
Regulatory Liaison	Map rules into checks	Approve policy templates	Risk and legal team
Observer Node	Monitor but not vote	Subscribe to finalized blocks	External partner

be Byzantine. A Byzantine node may deviate arbitrarily, including equivocating, forging messages under compromised keys, attempting to reorder transactions, or withholding votes. Honest nodes follow the protocol and do not leak private signing keys. We assume standard cryptographic hardness for digital signatures and hash functions. Network communication is authenticated and integrity-protected, though messages may be delayed, reordered, or dropped [10]. The network is partially synchronous: there exists an unknown global stabilization time after which message delays are bounded. This model is typical for Byzantine fault tolerant consensus and captures both transient partitions and variable latency in cross-region deployments.

Threats considered include tampering with governance records, tampering with lineage metadata, forging evidence, approving changes without required approvals, replaying old evidence for new models, and censorship of governance actions. Tampering with governance records is mitigated by consensus with quorum certificates and by requiring that finalized blocks are signed by a threshold of nodes. Tampering with lineage metadata is mitigated by content addressing and by committing lineage roots into finalized governance blocks. Forging evi-

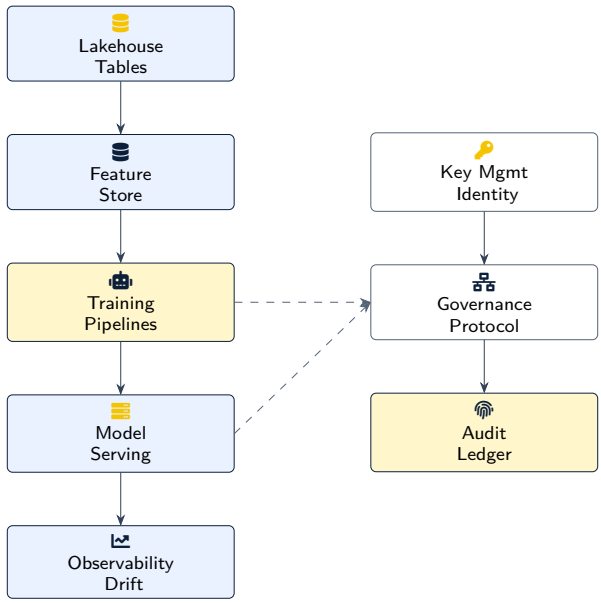


Figure 6. Integration of governance consensus into a data platform: core services (lakehouse, feature store, training, serving, observability) interact with a protocol sidecar anchored by identity/key management and an auditable ledger.

dence is mitigated by requiring evidence to be signed by authorized service principals and by binding evidence to specific artifact hashes and policy versions [11]. Replays are mitigated by nonces, epoch identifiers, and by requiring that evidence includes context fields referencing the relevant governance state. Censorship is mitigated by permitting clients to submit transactions to multiple nodes and by designing the mempool admission and proposal selection policies to be auditable, though the protocol acknowledges that complete censorship resistance is difficult in permissioned settings without additional economic or committee-rotation mechanisms.

The protocol also considers privacy and confidentiality constraints, which are common in governance. Auditors may be allowed to verify that required checks were performed without gaining access to raw datasets or proprietary model code. To accommodate this, evidence objects can include commitments to confidential artifacts and can be accompanied by selectively disclosed summaries. The protocol itself does not require that all evidence is publicly readable; it requires that evidence is addressable, integrity-protected, and verifiable under appropriate access controls [12]. Confidentiality is treated as orthogonal to consensus: the consensus layer commits to hashes and metadata, while access to underlying content is governed by platform access policies.

Operational constraints influence the design. Governance must not impose prohibitive latency on development iteration, particularly for early-stage experimentation, while still enforcing stricter controls on promotion to production. The protocol therefore distinguishes between governance stages, such as experimental, staging, and production, each associated with different admissibility predicates. The replicated state machine supports stage-specific policy evaluation and stage-specific authority requirements [13]. The design also anticipates offline or intermittent connectivity for some domains, which motivates mechanisms for delayed endorsement and for anchoring external attestations into the consensus log after the fact, while

Table 3. Consensus phases and message types in the networked protocol.

Phase	Message Type	Core Contents	Reliability Requirement
Proposal	`PROPOSE`	Change set, model id, lineage pointer	At least one honest proposer
Pre-vote	`PREVOTE`	Proposal hash, round number	$\geq 2f + 1$ matching votes
Pre-commit	`PRECOMMIT`	Locked proposal hash	$\geq 2f + 1$ matching commits
Finalize	`FINALIZE`	Block id, signatures, timestamp	Persisted to append-only log
Dispute	`CHALLENGE`	Evidence bundle, conflicting state	Delivered to all committee nodes

Table 4. Canonical lineage events captured in the audit log.

Event Type	Trigger	Recorded Fields	Example Query
Model Registered	New model created	Owner, schema hash, policy id	List all models owned by team X
Training Run	Training job completed	Data snapshot id, seed, metrics	Retrieve runs using dataset D7
Evaluation	Evaluation suite executed	Benchmark id, scores, threshold	Show failing fairness checks
Deployment	Model pushed to runtime	Target env, rollout plan, approver	Audit production deployments
Rollback	Deployment reverted	Previous version id, cause	Find all rollbacks in Q2
Retirement	Model decommissioned	End of life time, replacement	List retired models by domain

preserving causal consistency and preventing retroactive manipulation.

3. Protocol Design: Networked Consensus for Governance Actions

The protocol is structured as a permissioned Byzantine fault tolerant consensus system that replicates a deterministic governance state machine. Governance actions are expressed as transactions submitted by clients, validated by nodes against the current state, ordered through consensus, and applied to update the state. The consensus output is an append-only sequence of blocks, each block containing a set of transactions and a quorum certificate proving that a threshold of nodes agreed on the block. The governance state at height h is the result of deterministically applying all transactions from genesis through block h in order. This design enables any verifier with access to the finalized chain and the state transition rules to reconstruct the state, subject to access controls for off-chain content [14].

Transactions are typed and include structured fields that reference lineage and evidence. Representative transaction types include registering a dataset snapshot, registering a feature definition, registering a model artifact, submitting an evidence bundle, proposing a promotion of a model to a higher stage, approving or rejecting a proposal, updating a policy bundle,

Table 5. Core components in the data platform integration architecture.

Component	Purpose	Main Input	Main Output
Governance Node	Execute protocol and policies	Proposals, votes, policy rules	Finalized governance blocks
Model Registry	Track model versions	Model binaries, metadata	Stable model identifiers
Lineage Store	Persist history	Event records, signatures	Immutable lineage log
Policy Engine	Evaluate constraints	Policies, context, evidence	Pass / fail decision
Monitoring Bridge	Connect runtime signals	Telemetry, alerts	Governance-relevant events
Key Management Service	Secure identities	Key material, access rules	Rotated signing keys

Table 6. Security assumptions and corresponding mitigation mechanisms.

Assumption	Potential Risk	Mitigation Strategy
Bounded network delay	Liveness failure under network partitions	Timeouts with view changes and backoff
At most f Byzantine nodes	Colluding validators undermining votes	Stake-weighted committees and slashing
Honest majority of auditors	Suppressed or biased findings	Cross-organization audit redundancy
Immutable storage layer	Silent log tampering	Cryptographic commitments and cross-checks
Secure key custody	Key theft and impersonation	Hardware-backed keys and rotation policies

rotating keys, and changing authority assignments. Each transaction includes a unique identifier, a reference to the current governance epoch, a set of input references, and a set of signatures from required principals. Input references include hashes of artifacts, hashes of lineage nodes, and identifiers of existing registry objects. The uniqueness identifier prevents replay. Epoch references prevent cross-epoch reuse of approvals when policy changes occur [15]. Signatures ensure non-repudiation and enforce that only authorized principals can perform certain actions.

A key design element is the separation of proposal transactions from endorsement transactions, with both being recorded in the log. For example, promoting a model to production is represented by a proposal transaction that references the model version, the intended deployment target, the policy bundle to be applied, and the required evidence categories. The proposal transaction does not itself finalize the promotion. Endorsement transactions record approvals by reviewers, risk owners, and automated services, each endorsement including a signature and a reference to the proposal identifier. A finalization transaction, which can be automatically generated once endorsement conditions are met, causes the state machine to mark the promotion as finalized and to trigger downstream actions such as allowing deployment controllers to proceed [16]. This structure provides granular auditability: auditors can

Table 7. End-to-end performance of the protocol under increasing network size.

Nodes (n)	Median Latency (ms)	Throughput (tx/s)	Audit Overhead (%)
8	35	950	4.2
16	48	870	5.0
32	71	760	6.3
64	112	640	8.1
96	154	520	9.4
128	203	430	11.2

Table 8. Representative failure scenarios and how the system recovers from them.

Failure Mode	Detection Mechanism	Recovery Action
Validator crash	Missed heartbeats and votes	Reassign to standby node and resync state
Network partition	Divergent round numbers	Trigger view change and reconcile longest chain
Equivocating proposer	Conflicting proposals in same round	Slash stake and blacklist key material
Policy engine outage	Missing validation results	Buffer proposals and replay once restored
Corrupted lineage segment	Merkle proof mismatch	Restore from replicated snapshot

see who endorsed what, under which policy, with which evidence, and at which time in the canonical log.

Consensus nodes maintain a mempool of pending transactions. Admission control checks syntactic validity, signature correctness, nonce uniqueness, and basic authorization constraints. Deeper semantic checks, such as evaluating whether a proposal already has sufficient endorsements, are performed during execution when the transaction is applied in the consensus order. This prevents mempool divergence from affecting safety, since only the consensus-ordered execution determines state. Nodes may still choose to pre-filter transactions that are clearly invalid under the current state to reduce load, but must be prepared to handle transactions that become valid later due to state changes, such as endorsements arriving after a proposal is recorded [17].

The consensus protocol can be instantiated with a quorum-based algorithm that provides deterministic finality under the condition $n \geq 3f + 1$. Blocks are proposed by a leader that may rotate per round or per height. Nodes vote on proposed blocks after validating their contents and their linkage to the previous block. A block is considered finalized when it has a quorum certificate consisting of signatures from at least $2f + 1$ nodes. To reduce signature overhead, threshold signatures or aggregated signatures can be used, producing a compact quorum certificate that verifies against a group public key. Leader rotation and view-change mechanisms ensure liveness when a leader is faulty or slow [18]. The protocol assumes partial synchrony, so liveness is guaranteed after stabilization if at most f nodes are Byzantine.

Governance-specific constraints are integrated into block validation. Nodes validate that each transaction references a known epoch, that policy updates occur through authorized actions,

Table 9. Alignment between governed artifacts and required metadata in the platform.

Artifact Type	Key Metadata Fields	Governance Checks	Example Instance
Training Dataset	Source, license, pii status	Policy compliance, consent scope	Public web corpus, non-PII
Feature Store Table	Owner, refresh cadence	Drift monitoring, access control	User engagement aggregates
Model Binary	Framework, version, hash	Reproducibility, benchmark thresholds	Gradient boosted trees v2
Evaluation Suite	Metrics, datasets	Coverage, fairness, robustness	Latency and bias checks
Deployment Config	Region, rollout strategy	Residency, redundancy, rollback plan	Blue-green rollout in EU

and that any evidence references are well-formed and correspond to content-addressed objects. For transactions that reference lineage nodes, nodes validate that the referenced lineage node hashes are consistent with the claimed parent relationships, to the extent that the nodes have access to the lineage metadata. When full lineage content is not available to all nodes due to confidentiality, the protocol allows transactions to include a lineage root commitment and an access capability reference, enabling nodes to validate structural properties without reading confidential payloads. In such cases, validation relies on commitment schemes and signed attestations from trusted lineage services, and the governance state records which attestations were used [19].

A governance epoch is introduced as a mechanism to bind approvals and evidence to a particular policy context. Epoch changes occur when policy bundles change or when authority assignments are updated. Each epoch defines a policy version, an authority map, and a set of admissibility rules. Transactions include the epoch identifier, and endorsements must match the epoch of the proposal they endorse. This avoids ambiguity where a model might be proposed under one policy and then endorsed under another after a policy update. If a policy update is intended to affect in-flight proposals, the state machine can define deterministic migration rules, such as requiring re-endorsement under the new epoch [20]. The epoch mechanism also supports committee rotation in the consensus layer, enabling the set of consensus nodes to change over time while preserving audit continuity.

The protocol defines deterministic conflict resolution rules for concurrent actions. Because all actions are serialized by consensus, conflicts are resolved by order. However, the state machine must define how to handle semantic conflicts, such as two proposals attempting to reserve the same model alias or two key-rotation transactions targeting the same principal. Deterministic rules might include rejecting the later transaction if the resource is already reserved, or allowing both but marking one as superseded. The choice affects usability and audit semantics [21]. The design favors explicit supersession records rather than silent overwrites, since supersession preserves a clear lineage of decisions. This aligns with governance needs, where understanding why something changed is often as important as the final state.

Interoperation with external services is mediated through signed receipts and deterministic triggers. For example, a deployment controller may subscribe to finalized promotion events and then deploy the referenced model artifact. To ensure auditability, the deployment controller issues a signed deployment receipt transaction that records the deployed artifact hash, the target environment, the configuration hash, and the observed rollout status. Monitor-

ing systems can similarly submit signed incident and metric attestations that bind observed behavior to a deployed version [22]. These feedback signals become part of the governance state and lineage, supporting post-deployment audits and model retirement decisions. The protocol thus treats governance as a continuous lifecycle rather than a one-time approval gate.

4. Auditable Lineage: Content-Addressed Provenance Graphs Bound to Governance State

Auditable lineage requires more than storing identifiers that point to artifacts. It requires representing the causal and transformational relationships among artifacts in a way that is verifiable, composable, and resilient to partial disclosure. The protocol adopts a lineage substrate based on a content-addressed Merkle-DAG, where each lineage node is a structured record whose identifier is the cryptographic hash of its canonical serialization. A lineage node includes a type tag, a set of parent references, and a payload that may include metadata fields such as timestamps, executor identifiers, dataset logical names, schema digests, transformation code digests, and environment configuration digests [23]. Parent references create a directed acyclic graph capturing how outputs depend on inputs. For example, a training output node references parents corresponding to the dataset snapshot node, feature definition nodes, the code package node, and the container image node.

Content addressing provides tamper evidence: if any field of a node changes, its hash changes, and thus any governance transaction that committed to the old hash no longer matches. This property is central to audit. However, lineage graphs can be large, and not all participants need or are allowed to retrieve all nodes. The protocol therefore uses root commitments [24]. A governance transaction may include a lineage root hash that represents the transitive closure of dependencies relevant to a governance action, along with a proof object that enables a verifier to check inclusion of specific nodes. Proofs can be standard Merkle inclusion proofs when the DAG is normalized into a set commitment, or they can be path proofs that show how a particular node is reachable from the root through parent links. The protocol does not require a single global root for all lineage; it requires that each governance decision commits to the relevant root(s) and that the mapping from roots to concrete nodes is verifiable.

Evidence objects are treated as first-class nodes in the lineage graph. An evidence node includes the hash of the artifact being attested to, the policy predicate being evaluated, the measurement method, the measured result, and the signer identity [25]. For example, an evaluation evidence node might include a digest of the evaluation dataset snapshot, a digest of the evaluation code, and a digest of the produced metrics report, as well as a signed statement by the evaluation service principal. By representing evidence as lineage, the protocol allows auditors to traverse from a governance approval back through evidence to the exact pipeline and data that produced the evidence. This reduces ambiguity and helps prevent a common audit failure mode where an approval references a report by name but the report is later regenerated or replaced.

Governance state binds to lineage through commitments recorded in the consensus log. When a model version is registered, the registration transaction includes the model artifact hash and at least one lineage root hash representing the training run. When a promotion is proposed, the proposal includes the lineage root(s) required by policy, such as training lineage and evaluation lineage [26]. Endorsements include references to specific evidence nodes, and the finalization transaction commits to the set of endorsements and evidence nodes that satisfied the policy. This yields an auditable package: the finalized governance event includes all hashes necessary to reconstruct the decision and the underlying lineage graph. If auditors

later retrieve the content-addressed nodes, they can verify integrity by recomputing hashes and checking references. If content is unavailable due to retention limits, the protocol still preserves commitments that show what existed at the time of decision, though completeness then depends on archival practices.

A practical challenge is that many lineage-relevant artifacts are generated by complex workflows with nondeterministic elements, such as randomized training or environment-dependent behavior. The protocol does not attempt to make training deterministic [27]. Instead, it aims to make the description of what was executed and what was produced verifiable. This includes committing to the code digest, container digest, dependency lockfile digest, configuration digest, and data snapshot digest. For stochastic processes, the protocol can optionally record random seeds and library versions as part of the configuration node. The audit question becomes whether the recorded artifacts correspond to what was actually executed, which is addressed through signed receipts from trusted execution environments or from pipeline orchestrators that have privileged access to runtime information. The design acknowledges that such attestations rely on the integrity of the execution service and may require hardware-rooted trust for stronger guarantees.

Lineage also supports governance semantics such as data usage constraints and privacy requirements [28]. Dataset nodes can include usage policy tags, consent constraints, and retention classes. Feature nodes can include sensitivity classifications. Model nodes can include intended use declarations and risk categories. Policy predicates can then be defined over lineage attributes, for example requiring that production models do not depend on datasets with restricted consent tags, or requiring additional review if sensitive features are present. Because such attributes are part of the committed lineage nodes, policy evaluation can be audited: auditors can verify that the policy predicate was applied to the exact lineage attributes at the time of approval. When attributes themselves are sensitive, the protocol can store commitments to attributes and require that authorized auditors can reveal them later [29]. This introduces a trade-off between immediate verifiability by all nodes and confidentiality, which is addressed by specifying which predicates can be evaluated from commitments alone and which require privileged disclosure.

To manage scale, lineage nodes are grouped into shards and referenced by manifest nodes. A manifest node commits to an ordered list of child node hashes and can represent, for example, all partitions of a dataset snapshot or all intermediate steps of a pipeline. Governance transactions can then commit to manifest hashes rather than enumerating all children. Auditors can expand manifests as needed. This structure also supports incremental lineage: when a new model version is produced, the lineage graph can reuse existing nodes for unchanged components, such as a dataset snapshot or a feature definition, and only introduce new nodes for new outputs [30]. Content addressing naturally deduplicates identical nodes. The protocols storage model and audit cost depend on the degree of reuse, which is influenced by development practices such as versioning datasets by snapshot rather than by logical name.

A further consideration is lineage evolution. Organizations often refine how they represent lineage over time. The protocol supports schema evolution for lineage nodes by including explicit type tags and version fields in node serialization. Governance policy can then specify which lineage schema versions are acceptable for certain stages [31]. For example, experimental models might allow older lineage schema that lacks certain fields, while production promotion might require a schema version that includes explicit data consent metadata. By recording schema version in the node hash, the protocol prevents ambiguous interpretation and allows auditors to apply the correct parser and validation rules for historical data.

5. Formal Analysis and Mathematical Modeling of Correctness and Performance

The protocols correctness is analyzed in terms of consensus safety and liveness, governance admissibility, and audit integrity. Safety of the replicated log is inherited from the underlying Byzantine fault tolerant consensus algorithm under the assumption that $n \geq 3f + 1$ and that cryptographic primitives are unbroken. Safety can be expressed as agreement on the finalized sequence of blocks: no two honest nodes finalize different blocks at the same height. Given deterministic state transitions, this implies agreement on governance state [32]. Let B_h denote the finalized block at height h and let S_h denote the governance state after applying blocks through h. Safety implies that for any two honest nodes i and j, $B_h^i = B_h^j$ and therefore $S_h^i = S_h^j$ for all finalized heights. The protocols governance semantics are defined to be deterministic, including conflict resolution rules and policy evaluation, which is essential because nondeterminism would allow diverging states even under identical logs.

Liveness is the property that valid transactions submitted by honest clients are eventually included and finalized, assuming network conditions eventually stabilize and the number of Byzantine nodes does not exceed f. Liveness depends on leader rotation and view-change, and on the absence of permanent censorship by a coalition exceeding the fault threshold. In permissioned settings, membership and authority are governed by policy, so a separate governance question arises: who controls the consensus node set [33]. The protocol models membership changes as epoch transitions that update the validator set. Liveness then requires that epoch transitions themselves are subject to rules that prevent a Byzantine minority from freezing the system, such as requiring quorum certificates from the current epoch to authorize the next epoch. Under partial synchrony, standard arguments show that eventually an honest leader will propose a block and gather votes. The protocols added transaction validation rules must be designed not to introduce circular dependencies that block progress. For example, if a policy update requires evidence that itself requires policy update acceptance, the system could deadlock. The state machine design avoids such cycles by defining a base set of always-admissible maintenance actions and by allowing policy updates to be proposed and endorsed using existing policy until finalized [34].

Governance admissibility is captured by predicates evaluated by the state machine. Let P denote the current policy bundle and let T be a transaction. Define $\mathrm{Valid}(S, T)$ as a predicate that is true if and only if transaction T is admissible in state S under policy bundle P and authority mapping. For example, a promotion finalization transaction is valid only if the corresponding proposal exists, has not been finalized, and has accumulated endorsements satisfying a required threshold across roles, and if each required evidence category has at least one evidence node that is valid under P. Evidence validity can include signature checks and binding checks. Binding checks ensure that evidence references the correct artifact hashes and the correct policy epoch. A useful modeling approach treats endorsements and evidence as resources that satisfy policy constraints, and finalization consumes these resources in a deterministic way, recording the exact set consumed for audit [35].

Audit integrity is modeled by commitments. Let $H(\cdot)$ denote a collision-resistant hash function. Each lineage node x has an identifier $h_x = H(\mathrm{canon}(x))$, where $\mathrm{canon}(x)$ is a canonical serialization. A governance transaction commits to a set of lineage identifiers $\{h_{x_k}\}$ and evidence identifiers $\{h_{e_\ell}\}$. A finalized block commits to the ordered list of transactions and therefore transitively to those identifiers. For an auditor to verify that a governance decision was based on specific artifacts, the auditor must retrieve the corresponding nodes and check that recomputed hashes match committed hashes. Under collision resistance, an adversary

cannot substitute different content with the same hash except with negligible probability. Thus, audit integrity reduces to the availability of content and to the trustworthiness of signers for evidence, rather than to the mutability of logs [36].

The protocols performance can be modeled by decomposing end-to-end governance latency into consensus latency and evidence availability latency. Let Δ denote the network delay bound after stabilization, and let r denote the number of consensus rounds required for finalization in the chosen algorithm. In optimistic conditions, a block can be finalized in a small constant number of message delays, often approximated as $O(\Delta)$. The effective governance latency for a transaction that requires endorsements depends on the time to gather endorsements from human and automated principals, which is typically dominated by external processes. However, the protocols overhead should not materially add to this [37]. The added overhead includes hashing and signature verification, which can be quantified by per-transaction CPU cost, and state storage and retrieval cost, which can be quantified by database operations.

A simplified throughput model can be derived by considering block size and validation cost. Let λ be the transaction arrival rate, let b be the block interval, and let m be the maximum number of transactions per block. The system can sustain λ up to approximately m/b under steady state, subject to CPU and I/O constraints. Validation cost per transaction includes signature verification cost c_σ, hash verification cost c_h, and state access cost c_s. If a node has CPU budget C per second for validation, then a rough constraint is $\lambda(c_\sigma + c_h + c_s) \leq C$ [38]. Aggregated signatures reduce the per-block overhead by replacing many signature checks with fewer, though transaction signatures still need verification unless transactions are also aggregated via batch verification. If each transaction includes k signatures on average, then c_σ scales with k, which motivates role-based threshold endorsements where possible.

Storage growth is driven by the consensus log and by the lineage substrate. Let g be the average governance transaction size in bytes and let λ be the average transaction rate. Then the log grows at approximately λg bytes per second. Lineage growth depends on the number of new lineage nodes per model lifecycle event and on deduplication [39]. Let u be the average number of unique lineage nodes introduced per registered model version, and let s be the average size of a lineage node. If model versions are registered at rate ρ, then lineage storage grows at ρus. Evidence bundles, such as evaluation reports, can dominate storage if stored in full. In the protocol, evidence payloads are stored off-chain and referenced by hash, so the consensus log contains only commitments. Nevertheless, auditors may require retention, so the platform must manage archival storage. A governance-aware retention model can be expressed as a function of stage and risk category, with higher-risk production artifacts retained longer [40].

Consensus network bandwidth can be modeled by message complexity. Many BFT protocols require $O(n^2)$ message exchange per round, though optimized variants reduce practical overhead through leader-based aggregation. If each vote message is size v bytes and each node sends votes to the leader, then the leader receives roughly nv per round and broadcasts a proposal or certificate of size p bytes. Total network load is influenced by block size and by quorum certificate size. Threshold signatures reduce certificate size to a constant, which is useful when n is large. However, threshold signing requires distributed key generation or a trusted setup, and key rotation becomes more complex [41]. The protocol allows either approach, treating it as an implementation choice.

Governance correctness also involves reasoning about the binding between policy evaluation and evidence. Let E denote an evidence node and let a denote the artifact hash being attested.

A binding predicate $\mathrm{Bind}(E, a)$ is true if E contains a and any required contextual fields such as dataset snapshot hash and code hash. Let $\mathrm{Auth}(E)$ be true if the signer of E is authorized for the evidence category under the current authority mapping. Let $\mathrm{Fresh}(E, \epsilon)$ be true if E is issued within an allowed window relative to epoch ϵ and references ϵ explicitly. Evidence validity can be defined as

$$\mathrm{EvValid}(S, E, a, \epsilon) \;=\; \mathrm{SigOk}(E) \;\wedge\; \mathrm{Auth}(E) \;\wedge\; \mathrm{Bind}(E, a) \;\wedge\; \mathrm{Fresh}(E, \epsilon) \quad , \tag{5.1}$$

where $\mathrm{SigOk}(E)$ checks the digital signature and identity binding. A policy predicate for promotion can then be expressed as a conjunction of required evidence categories and required endorsement roles. If $\mathcal{C}$ is the set of evidence categories required and $\mathcal{R}$ is the set of roles required, the predicate can be modeled as

$$\mathrm{PromoteOk}(S, \mathrm{prop}) \;=\; \Big(\underset{c \in \mathcal{C}}{} \exists E \in \mathrm{Ev}(\mathrm{prop}, c) \;:\; \mathrm{EvValid}(S, E, a, \epsilon) \Big) \;\wedge\; \Big(\underset{r \in \mathcal{R}}{} \exists \pi \in \mathrm{End}(\mathrm{prop}, r) \;:\; \mathrm{En} \tag{5.2}$$

where $\mathrm{Ev}(\mathrm{prop}, c)$ is the set of evidence nodes associated with proposal prop for category c, $\mathrm{End}(\mathrm{prop}, r)$ is the set of endorsements for role r, a is the model artifact hash, and ϵ is the proposal epoch. Determinism requires that the state machine selects a canonical witness set of evidence and endorsements when multiple satisfy the predicate, for example by choosing the lexicographically smallest hashes [42]. This canonicalization ensures that finalization records the same set on all nodes.

An adversarial analysis focuses on attempts to violate PromoteOk while still finalizing. Because finalization is executed deterministically by honest nodes, an adversary would need to either compromise signatures for required roles, or exploit a validation weakness where EvValid accepts incorrect bindings, or cause divergent interpretations of policy. The design mitigates this by requiring that policies themselves are part of the replicated state, with canonical serialization and hashing. Policy bundles are committed in blocks and referenced by epoch identifiers. Therefore, policy interpretation is uniform across nodes. Another attack vector is submitting evidence that is technically valid but semantically misleading, such as evaluating on a favorable dataset snapshot. The protocol can address this only to the extent that policy constrains which datasets are acceptable and requires disclosure of dataset hashes [43]. Ultimately, governance captures accountability and traceability rather than guaranteeing that a model is appropriate in an absolute sense.

A final modeling aspect concerns audit cost. Suppose an auditor wants to verify a finalized promotion event. The auditor must retrieve the promotion finalization record, the associated proposal, the endorsements, and the evidence nodes. The auditor must then fetch the lineage roots and expand the lineage DAG to a degree required by audit scope. Let N be the number of lineage nodes reachable from the committed root, and let q be the fraction the auditor chooses to inspect in depth [44]. The audit retrieval cost is proportional to qN, while integrity verification cost is proportional to the number of hashes recomputed and signatures checked. If each node requires one hash computation and each evidence requires one signature verification, then computational cost can be approximated as $qNc_h + Mc_\sigma$, where M is the number of evidence and endorsement signatures. The protocol supports partial audit by allowing auditors to verify inclusion of specific nodes via proofs without fetching all nodes, reducing q for routine audits while enabling deep audits when needed.

6. Implementation Considerations in Data Platforms and Evaluation Methodology

Implementing the protocol in a real data platform requires careful integration with existing services while preserving determinism and auditability. The consensus layer can be deployed as a set of governance nodes operated by different teams or domains. Each node maintains the replicated log and a local database for state indexing [45]. The log itself can be stored in an append-only store with periodic checkpoints. Checkpoints allow nodes to bootstrap without replaying the entire history, while still enabling auditors to verify history if needed. Checkpointing must be done carefully to preserve audit semantics: a checkpoint should include the state hash and a proof linking it to a finalized block, so that the checkpoint is not a new trust anchor but a derived artifact. Nodes can maintain both a full archive mode and a pruned mode, where pruned nodes rely on checkpoints and retain recent blocks, while archival nodes retain the full chain.

The lineage substrate can be implemented as a content-addressed store backed by object storage, with a canonical serialization format for nodes. A practical choice is a binary canonical form that avoids ambiguity in field ordering and encoding [46]. The platforms existing metadata store can be used as an index from logical names to content hashes, but the authoritative identity of a lineage node remains its hash. The pipeline orchestration system should be extended to emit lineage nodes as part of execution, including input dataset snapshot hashes, code hashes, environment hashes, and output artifact hashes. This emission should occur in a way that is resilient to retries: if the same execution is retried, it should produce the same lineage nodes for identical inputs, or produce a new execution node that references the previous attempt, depending on the desired semantics. Deterministic naming of execution nodes can be achieved by hashing a tuple of inputs, though runtime nondeterminism may still affect outputs, so the protocol records outputs explicitly.

Evidence services, such as vulnerability scanners and evaluation harnesses, should be modeled as service principals with managed keys. Each service produces signed evidence nodes and writes them to the content-addressed store [47]. To prevent key compromise from silently rewriting history, evidence issuance can be coupled with a transparency log: evidence nodes are committed to the governance log as soon as they are produced, or at least before they are used in approvals. This ensures that evidence used for approvals is part of the canonical record. For high-stakes stages, services can use hardware-backed keys or remote attestation to strengthen confidence that evidence was produced by an approved binary. The protocol treats such strengthening as optional but records the attestation mechanism in evidence metadata so auditors can judge assurance level.

Integration with a model registry involves representing model versions as registry objects in governance state and storing the actual artifacts in an artifact store. A model registration transaction includes the model artifact hash, a pointer to its storage location, and the lineage root hash of the training run [48]. The artifact store must provide immutability or versioned retention to ensure that a stored object at a given hash remains retrievable. If the artifact store is not naturally content-addressed, a wrapper can compute hashes on upload and enforce write-once semantics by mapping hashes to immutable object versions. The governance state can also store human-readable labels and aliases, but those are treated as mutable pointers that change through governance actions, never by overwriting an artifact.

Deployment integration is often the most sensitive operational point. Deployment controllers typically require high availability and low latency [49]. The protocol supports a pattern where deployment controllers treat the governance log as the source of truth for which model versions

are authorized for which environments. Controllers subscribe to finalized events and validate quorum certificates before acting. To avoid constant chain verification overhead, controllers can rely on checkpointed state snapshots signed by a quorum and periodically verify linkage to the chain. Deployment receipts are submitted back as governance transactions, creating a closed audit loop. This approach allows independent verification of what was authorized and what was deployed, reducing reliance on mutable deployment logs.

Policy authoring and evaluation must be deterministic [50]. Policies are represented as code or declarative rules with a canonical evaluation engine. In practice, this implies restricting policy languages to deterministic subsets and specifying exact evaluation semantics, including numeric precision and string normalization. If policies include numeric thresholds, evaluation must be consistent across platforms. For example, if a policy requires that a metric exceeds a threshold, the metric computation must be referenced by hash and the policy must specify how the metric is parsed and rounded. Evidence nodes should include raw values and canonical encodings. This reduces disputes where a metric appears to pass on one system and fail on another due to formatting differences [51]. The governance state records the policy engine version, enabling auditors to reproduce evaluation behavior.

Key management and identity federation are operationally complex. The protocol assumes that each principals key is bound to identity through signed assertions. Implementations should support key rotation by recording key update transactions that link old and new keys and specify effective times. For service principals, rotations should be automated and integrated with secret management. For human principals, rotations may be driven by employment changes or security events [52]. Revocation is modeled as a governance action that updates authority mappings. The protocols audit log then captures who had authority at each time, which is often required for compliance. In federated settings, mapping between local identities and global identifiers should be explicit and signed, and disputes should be handled by governance actions that record overrides or deprecations.

Evaluation of the protocol should emphasize correctness, performance, and audit utility. Correctness evaluation includes fault-injection tests where a subset of nodes behave Byzantine, including equivocation, vote withholding, invalid transaction proposals, and censorship attempts. The system should be tested for safety under these faults, demonstrating that honest nodes do not finalize conflicting states [53]. Liveness evaluation includes network delay and partition simulations, measuring time to finalize after stabilization. Governance semantics should be tested with adversarial transactions, such as replayed endorsements, evidence bound to wrong artifacts, and proposals that attempt to exploit policy edge cases. Deterministic state machine behavior should be validated by running multiple nodes with different hardware and operating systems and verifying identical state hashes for the same log.

Performance evaluation should measure consensus throughput and latency with realistic transaction mixes. Governance workloads are not uniform. Experimental stages may generate many registrations and evidence submissions, while production stages generate fewer but heavier approval workflows [54]. Measurements should include signature verification overhead, state database performance, and network bandwidth. Because evidence payloads are off-chain, the consensus layer primarily handles commitments and signatures, but some transactions may include multiple signatures, increasing cost. The effect of aggregated signatures should be quantified. Another performance dimension is audit verification time. Auditors may need to verify a set of promotions over a period. The system should provide efficient query paths to retrieve all relevant hashes and signatures, and it should support batch verification of signatures [55]. The relationship between lineage graph size and audit time should be measured, including the benefits of manifests and deduplication.

Audit utility evaluation should involve reconstructing lineage for a sample of deployed models and verifying that required evidence is present and correctly bound. The system should support both routine audits, where only a subset of lineage nodes is inspected, and deep audits, where full lineage is expanded. The protocols ability to support partial disclosure should be evaluated by simulating an auditor with restricted access and verifying that commitments still allow integrity checking. Retention and archival policies should be tested, ensuring that commitments remain verifiable even if some content is moved to cold storage. An important metric is audit completeness: the fraction of governance decisions for which all required evidence and lineage nodes remain available and verifiable [56]. Another metric is audit interpretability: the clarity with which the log and lineage graphs can be navigated to answer typical audit questions such as which dataset snapshots contributed to a model, which reviewers approved it, and which policy version applied.

The protocol also needs to coexist with human workflows. User interfaces should present governance state in a way that is understandable and that encourages correct behavior. For example, when a developer proposes a promotion, the interface can automatically assemble the required evidence set by querying the lineage store and evidence services, reducing manual attachment errors. Reviewers can be presented with a deterministic summary that includes hashes, policy checks, and evidence signatures, along with human-readable descriptions derived from metadata. The protocols deterministic core does not depend on the user interface, but usability affects whether the system is adopted and whether governance records are complete [57]. The design anticipates that some teams will operate in a mode where they create many experimental artifacts that are later garbage-collected, while production artifacts require strict retention. Stage-specific policies can encode these differences.

A final implementation consideration is interoperability across platforms. Organizations may use multiple data warehouses, multiple orchestration engines, and multiple registries. The protocol is designed to be a thin governance layer that can bind to heterogeneous artifact stores via content hashes. Adapters can compute hashes and produce lineage nodes for each system [58]. The replicated governance state serves as a unifying record that can reference artifacts regardless of where they are stored. This reduces the need to centralize all artifacts into a single system, which is often impractical. It also supports supply-chain scenarios where a vendor provides a model artifact and evidence under their own domain, and a consumer domain records acceptance and deployment under its own authority, with cross-domain lineage references captured as hash commitments and signed assertions.

7. Conclusion

This paper has described a networked consensus protocol for collaborative model governance in data platforms, with an emphasis on auditable lineage and deterministic, verifiable decision records. The design models governance as a replicated state machine whose transactions represent proposals, endorsements, evidence commitments, and policy updates, all ordered by a Byzantine fault tolerant consensus mechanism that provides deterministic finality under standard fault assumptions. Auditable lineage is represented as a content-addressed provenance graph, and governance decisions commit to lineage roots and evidence nodes, enabling independent reconstruction and verification of how a model was produced, evaluated, approved, and deployed [59]. The protocol integrates epoch-based policy contexts to bind approvals and evidence to specific policy versions, reducing ambiguity during policy evolution, and it records operational feedback such as deployment receipts and monitoring attestations to support lifecycle governance beyond initial approval.

The analysis has outlined how consensus safety and liveness interact with governance semantics, how evidence binding can be formalized to prevent replays and substitutions under realistic assumptions, and how performance and audit costs can be modeled in terms of transaction validation, signature overhead, storage growth, and lineage expansion. Implementation considerations highlight the need for deterministic policy evaluation, robust identity and key management, practical lineage emission from pipelines, and integration patterns that keep operational systems responsive while preserving auditability. The evaluation methodology emphasizes adversarial testing, reproducible audits, and measurements that reflect the distinct workload characteristics of governance compared to general-purpose transaction systems.

The protocol does not eliminate the need for organizational judgment about policies, risk thresholds, and trusted execution services, and it does not guarantee semantic correctness of evidence beyond what policies specify. It provides a structured way to record and verify governance decisions and their supporting lineage under collaborative and adversarial conditions, offering a basis for more consistent audits and for clearer accountability in complex model lifecycles [60].

References

[1] Boris Sedlak, Víctor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar. "Diffusing High-level SLO in Microservice Pipelines". In: *2024 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, July 15, 2024, pp. 11–19. DOI: 10.1109/sose62363.2024.00008.

[2] Randolf Rotta, Navya Sneha Chatharajupalli, Billy Naumann, Julius Schulz, Reinhardt Karnapke, Matthias Werner, and Jörg Nolte. "Demo: B.A.T.M.A.N. Mesh Routing on Ultra Low-Power IEEE 802.11 Modules". In: *2024 IEEE 49th Conference on Local Computer Networks (LCN)*. IEEE, Oct. 8, 2024, pp. 1–4. DOI: 10.1109/lcn60385.2024.10639789.

[3] Sam Ghosh. "Primer on Web3 and Distributed Systems". In: Productivity Press, Aug. 27, 2024, pp. 33–67. DOI: 10.4324/9781003507352-4.

[4] Jinchao Chen, Chenglie Du, Pengcheng Han, and Xiaoyan Du. "Real-time digital simulator for distributed systems:" in: *SIMULATION* 97.5 (Jan. 22, 2021), pp. 003754972098686–309. DOI: 10.1177/0037549720986865.

[5] Manfred Broy. *Concurrent Distributed Systems Beyond Monotonicity*. Jan. 1, 2024. DOI: 10.2139/ssrn.5057101.

[6] Jackson Raffety, Brooklynn Stone, Jan Svacina, Connor Woodahl, Tomas Cerny, and Pavel Tisnovsky. "Multi-source Log Clustering in Distributed Systems". In: Germany: Springer Singapore, Apr. 3, 2021, pp. 31–41. DOI: 10.1007/978-981-33-6385-4_4.

[7] Daszczuk W. *Modelling Distributed Systems in Distributed Autonomous and Asynchronous Automata (DA3)*. Dec. 31, 2019. DOI: 10.20944/preprints201912.0400.v1.

[8] Azkario Rizky Pratama, Frans Juanda Simanjuntak, Aliaksandr Lazovik, and Marco Aiello. *APPIS - Low-power Appliance Recognition using Recurrent Neural Networks*. Mar. 29, 2018.

[9] Rahul Malik, Sangkyum Kim, Xin Jin, Chandrasekar Ramachandran, Jiawei Han, Indranil Gupta, and Klara Nahrstedt. "MLR-index: An index structure for fast and scalable similarity search in high dimensions". In: *International Conference on Scientific and Statistical Database Management*. Springer. 2009, pp. 167–184.

[10] "DISTRIBUTED SYSTEM MODEL FOR KEY MANAGEMENT". In: *Bulletin of TUIT: Management and Communication Technologies* (Jan. 20, 2018), pp. 1–5. DOI: 10.51348/tuitmct115.

[11] "Message from the Program Chairs". In: *2023 42nd International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, Sept. 25, 2023, pp. x–x. DOI: 10.1109/srds60354.2023.00006.

[12] Ege Berkay Gulcan, Burcu Kulahcioglu Ozkan, Rupak Majumdar, and Srinidhi Nagendra. *Model-guided Fuzzing of Distributed Systems*. Oct. 3, 2024. DOI: 10.48550/arxiv.2410.02307.

[13] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. *Global Winning Conditions in Synthesis of Distributed Systems with Causal Memory*. July 20, 2021.

[14] Jiacheng Shi, Yang Yu, Jinyu Gu, and Yubin Xia. "A Hardware-Software Co-Design for Efficient Secure Containers". In: *Proceedings of the Twentieth European Conference on Computer Systems*. ACM, Mar. 30, 2025, pp. 1229–1245. DOI: 10.1145/3689031.3717473.

[15] Cristina Seceleanu. "ISoLA (3) - ISoLA 2018 - Verification and Validation of Distributed Systems: Track Introduction". In: Germany: Springer International Publishing, Oct. 31, 2018, pp. 323–326. DOI: 10.1007/978-3-030-03424-5_21.

[16] Mengmeng Sun, Yiping Teng, Fanyou Zhao, Jiawei Qi, Dongyue Jiang, and Chunlong Fan. "Spatio-Textual Group Skyline Query". In: Germany: Springer Nature Switzerland, Sept. 28, 2023, pp. 34–50. DOI: 10.1007/978-3-031-35415-1_3.

[17] Philipp Waibel, Christoph Hochreiner, Stefan Schulte, Agnes Koschmider, and Jan Mendling. "ViePEP-C: A Container-Based Elastic Process Platform". In: *IEEE Transactions on Cloud Computing* 9.4 (Oct. 1, 2021), pp. 1657–1674. DOI: 10.1109/tcc.2019.2912613.

[18] Jorge F. Herculano, Luís O. S. de Andrade, Willians De P. Pereira, and Alirio S. de Sá. "E-THeTA: An Efficient and Adaptive TDMA Approach for Wireless Body Area Networks". In: *2024 XIV Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE, Nov. 26, 2024, pp. 1–6. DOI: 10.1109/sbesc65055.2024.10771820.

[19] Fabian Ritz, Thomy Phan, Andreas Sedlmeier, Philipp Altmann, Jan Wieghardt, Reiner Schmid, Horst Sauer, Cornel Klein, Claudia Linnhoff-Popien, and Thomas Gabor. "Capturing Dependencies Within Machine Learning viaăaăFormal Process Model". In: Germany: Springer Nature Switzerland, Oct. 17, 2022, pp. 249–265. DOI: 10.1007/978-3-031-19759-8_16.

[20] N.I. Tsiporkov and N.N. Grinchenko. "DEVELOPMENT OF A DISTRIBUTED SYSTEM FOR AUTOMATION OF THE LEARNING PROCESS". In: *Modern Technologies in Science and Education MTSE-2020, Vol.3*. Ryazan State Radio Engineering University, June 15, 2020, pp. 142–145. DOI: 10.21667/978-5-6044782-1-9-142-145.

[21] Vivek Vijaykumar, R Chandrasekar, and T Srinivasan. "An obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs". In: *2006 IEEE Conference on Cybernetics and Intelligent Systems*. IEEE. 2006, pp. 1–6.

[22] Finn Hackett, Shayan Hosseini, Renato Costa, Matthew Do, and Ivan Beschastnikh. "Compiling Distributed System Models with PGo". In: *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ACM, Jan. 27, 2023, pp. 159–175. DOI: 10.1145/3575693.3575695.

[23] Girish Talmale and Urmila Shrawankar. "Cluster Based Real Time Scheduling for Distributed System". In: *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 10.2 (Mar. 19, 2021), pp. 137–156. DOI: 10.14201/adcaij2021102137156.

[24] Lauren Pick, Ankush Desai, and Aarti Gupta. "Psym: Efficient Symbolic Exploration of Distributed Systems". In: *Proceedings of the ACM on Programming Languages* 7.PLDI (June 6, 2023), pp. 660–685.

[25] Xiaofan Liu, Xunjin Zheng, Cai Fu, Xiaoheng Xie, and Peng Di. "GrayDuck: The Sword of Damocles for Duck Typing in Dynamic Language Deserialization". In: *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. ACM, Oct. 27, 2024, pp. 2458–2459. DOI: 10.1145/3691620.3695327.

[26] Andrei A. Klishin, David J. Singer, and Greg van Anders. "Avoidance, adjacency, and association in distributed systems design". In: *Journal of Physics: Complexity* 2.2 (Apr. 8, 2021), pp. 025015–. DOI: 10.1088/2632-072x/abe27f.

[27] Geoffrey Neumann, Paul Grace, Daniel Burns, and Mike Surridge. "Pseudonymization risk analysis in distributed systems". In: *Journal of Internet Services and Applications* 10.1 (Jan. 8, 2019), pp. 1–16. DOI: 10.1186/s13174-018-0098-z.

[28] Reem M. Albarrak and Daniel A. Menasce. "Trust But Verify: A Framework for the Trustworthiness of Distributed Systems". In: *IEEE Transactions on Dependable and Secure Computing* 19.3 (May 1, 2022), pp. 2105–2121. DOI: 10.1109/tdsc.2020.3048301.

[29] Wenjun Wu, Zhuowei Shen, and Defeng Wang. "A Running Pattern Recognition Algorithm for Publish/Subscribe Distributed Systems". In: *DEStech Transactions on Materials Science and Engineering* ameme (Apr. 7, 2021). DOI: 10.12783/dtmse/ameme2020/35562.

[30] Daniel Pfleger and Ulrich Schmid. *SIROCCO - On Knowledge and Communication Complexity in Distributed Systems*. Germany, Oct. 31, 2018. DOI: 10.1007/978-3-030-01325-7_27.

[31] Francesc D. Muñoz-Escoí and Rubén de Juan-Marín. "On synchrony in dynamic distributed systems". In: *Open Computer Science* 8.1 (Aug. 1, 2018), pp. 154–164. DOI: 10.1515/comp-2018-0014.

[32] *IEEE Distributed Systems Online*. Mar. 21, 2023. DOI: 10.1109/mdso.8968.

[33] R Chandrasekar and T Srinivasan. "An improved probabilistic ant based clustering for distributed databases". In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*. 2007, pp. 2701–2706.

[34] Kwanwai Mak, Koichi Osuka, and Teruyo Wada. "Development of a Multi-Master Communication Platform for Mobile Distributed Systems". In: *Journal of Robotics and Mechatronics* 31.2 (Apr. 20, 2019), pp. 348–354. DOI: 10.20965/jrm.2019.p0348.

[35] Nor Syazwani Mohd Pakhrudin, Murizah Kassim, and Azlina Idris. "A review on orchestration distributed systems for IoT smart services in fog computing". In: *International Journal of Electrical and Computer Engineering (IJECE)* 11.2 (Apr. 1, 2021), pp. 1812–1822. DOI: 10.11591/ijece.v11i2.pp1812-1822.

[36] Andrey Polyakov. *On global existence of strong and classical solutions of Navier-Stokes equations*. Apr. 11, 2019.

[37] Roman Matzutt, Vincent Ahlrichs, Jan Pennekamp, Roman Karwacik, and Klaus Wehrle. "A Moderation Framework for the Swift and Transparent Removal of Illicit Blockchain Content". In: *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, May 2, 2022, pp. 1–9. DOI: 10.1109/icbc54727.2022.9805508.

[38] Youness Braidiz, Denis Efimov, Andrey Polyakov, and Wilfrid Perruquetti. "On robustness of finite-time stability of homogeneous affine nonlinear systems and cascade interconnections". In: *International Journal of Control* (Sept. 9, 2020), pp. 1–11.

[39] T Srinivasan, R Chandrasekar, Vivek Vijaykumar, V Mahadevan, A Meyyappan, and A Manikandan. "Localized Tree Change Multicast Protocol for Mobile Ad Hoc Networks". In: *2006 International Conference on Wireless and Mobile Communications (ICWMC'06)*. IEEE. 2006, pp. 44–44.

[40] Dimitrios Vasilas. *Search on Secondary Attributes in Geo-Distributed Systems*. Jan. 1, 2018. DOI: 10.48550/arxiv.1801.02974.

[41] Yixiao Wang. "A Distributed System Fault Diagnosis System based on Machine Learning". In: *Scalable Computing: Practice and Experience* 25.2 (Feb. 24, 2024), pp. 1117–1123. DOI: 10.12694/scpe.v25i2.2622.

[42] Victor Casamayor Pujol, Boris Sedlak, Yanwei Xu, Praveen Kumar Donta, and Schahram Dustdar. "Invited Paper: DeepSLOs for the Computing Continuum". In: *Proceedings of the 2024 Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*. ACM, June 17, 2024, pp. 1–10. DOI: 10.1145/3663338.3663681.

[43] Nitin Naik. "ISSE - Comprehending Concurrency and Consistency in Distributed Systems". In: *2021 IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, Sept. 13, 2021, pp. 1–6. DOI: 10.1109/isse51541.2021.9582518.

[44] Ala Arman, Pierfrancesco Bellini, Daniele Bologna, Paolo Nesi, Gianni Pantaleo, and Michela Paolucci. "Automating IoT Data Ingestion Enabling Visual Representation." In: *Sensors (Basel, Switzerland)* 21.24 (Dec. 17, 2021), pp. 8429–8429. DOI: 10.3390/s21248429.

[45] R Chandrasekar, RK Suresh, and SG Ponnambalam. "Evaluating an obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs". In: *2006 International Conference on Advanced Computing and Communications*. IEEE. 2006, pp. 628–629.

[46] Esmat Edres Mohammed, Rafat Yousef Saeed Naji, Ahmed Abdulsalam Hussein, Mogeeb A. Saeed, and Raad Abdo Mohammed Al Selwi. "Anomaly Detection System For Secure Cloud Computing Environment Using Machine Learning". In: *2025 5th International Conference on Emerging Smart Technologies and Applications (eSmarTA)*. IEEE, Aug. 5, 2025, pp. 1–9. DOI: 10.1109/esmarta66764.2025.11132254.

[47] Jakub Jaros, David Hruby, Vladimir Vasinek, and Tomas Stratil. "Measuring of the petroleum product leaks by distributed systems". In: *Fourth International Conference on Photonics Solutions (ICPS2019)*. SPIE, Mar. 11, 2020, pp. 35–35. DOI: 10.1117/12.2554423.

[48] Ammad Ul Islam, Ming Yang, Amna Iqbal, Anqi Xu, Sijuan Chen, and Sohail Asghar. "Trustworthy Distributed Systems: A Machine Learning Approach". In: *2025 IEEE 20th Conference on Industrial Electronics and Applications (ICIEA)*. IEEE, Aug. 3, 2025, pp. 1–6. DOI: 10.1109/iciea65512.2025.11148715.

[49] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. *Global Winning Conditions in Synthesis of Distributed Systems with Causal Memory (Full Version)*. Oct. 18, 2021.

[50] Deepthi Vishwanath. *Validating Key-value Based Implementations of the Raft Consensus Algorithm for Distributed Systems*. Apr. 1, 2018. DOI: 10.31979/etd.mjrc-2apu.

[51] Alfreds Lapkovskis, Boris Sedlak, Sindri Magnússon, Schahram Dustdar, and Praveen Kumar Donta. "Benchmarking Dynamic SLO Compliance in Distributed Computing Continuum Systems". In: *2025 IEEE International Conference on Edge Computing and Communications (EDGE)*. IEEE, July 7, 2025, pp. 93–102. DOI: 10.1109/edge67623.2025.00020.

[52] Yun Qian, Shi Ying, and Bingming Wang. "SMC - Anomaly Detection in Distributed Systems via Variational Autoencoders". In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, Oct. 11, 2020, pp. 2822–2829. DOI: 10.1109/smc42975.2020.9283078.

[53] *Coordination in constrained and natural distributed systems*. June 1, 2019. DOI: 10.3030/840605.

[54] Oluwadamilola Okusanya. *Consensus in Distributed Systems: RAFT vs CRDTs*. May 28, 2019.

[55] Danil I. Sukhoplyuev and Alexey N. Nazarov. “Monitoring fault tolerance in distributed systems”. In: *Computational nanotechnology* 11.4 (Sept. 24, 2024), pp. 94–106. DOI: 10.33693/2313-223x-2024-11-4-94-106.

[56] Sebastian Bindick, Hendrik Kausch, Mathias Pfeiffer, Deni Raco, Amelie Rath, Bernhard Rumpe, and Andreas Schweiger. “Communication and Architectural Patterns for Developing Distributed Systems”. In: *2025 8th International Conference on Software and System Engineering (ICoSSE)*. IEEE, Apr. 4, 2025, pp. 31–38. DOI: 10.1109/icosse65712.2025.00014.

[57] Rahul Malik, Chandrasekar Ramachandran, Indranil Gupta, and Klara Nahrstedt. “Samera: a scalable and memory-efficient feature extraction algorithm for short 3D video segments.” In: *IMMERSCOM*. 2009, p. 18.

[58] Fangnuo Wu, Mingkai Dong, Gequan Mo, and Haibo Chen. “TreeSLS: A Whole-system Persistent Microkernel with Tree-structured State Checkpoint on NVM”. In: *Proceedings of the 29th Symposium on Operating Systems Principles*. ACM, Oct. 23, 2023, pp. 1–16. DOI: 10.1145/3600006.3613160.

[59] Nezha Benmoussa, Abir Elyamami, Khalifa Mansouri, Mohammed Qbadou, and Elhoussein Illoussamen. “A Multi-Criteria Decision Making Approach for Enhancing University Accreditation Process”. In: *Zenodo (CERN European Organization for Nuclear Research)* (Feb. 16, 2019). DOI: 10.5281/zenodo.2576258.

[60] Hanan Shukur, Subhi Zeebaree, Rizgar Zebari, Omar Ahmed, Lailan Haji, and Dildar Abdulqader. “Cache Coherence Protocols in Distributed Systems”. In: *Journal of Applied Science and Technology Trends* 1.2 (June 24, 2020), pp. 92–97. DOI: 10.38094/jastt1329.