A framework for AI lineage. Timestamp Integrity, and Compute Provenance Anchored to the Kaspa DAG.

Author; 12GaugeKenshin
Version; 0.1
Date 11/13/2025

Abstract

Modern AI systems operate without verifiable integrity: models can fabricate results, misrepresent when computations occurred, or alter lineage without detection. Existing provenance approaches rely on centralized infrastructure, mutable logs, or unverifiable metadata. They provide no cryptographic guarantee that a model actually executed a computation, no resistance to backdating or replay attacks, and no mechanism for reconstructing trustworthy model lineage. As AI systems increasingly influence financial markets, governance, healthcare, and national-security domains, the inability to validate compute origin and timing represents a structural vulnerability.

This paper presents **Proof of Compute Integrity (PoCI)** - a framework for converting AI compute events into signed, timestamp-anchored commitments suitable for decentralized verification. PoCI introduces a **four-loop architecture** that separates (1) event creation, (2) proof generation, (3) on-chain anchoring, and (4) lineage verification. This structure provides a practical path toward accountable AI without exposing model parameters, private data, or full inference traces. An **adaptive integrity controller** dynamically adjusts verification requirements based on observed model behavior, aligning incentives with learnability-equilibrium principles and mitigating long-term adversarial drift.

PoCI anchors proofs to the **Kaspa PoW blockDAG**, leveraging its parallel block-processing, predictable finality, and physics-anchored timestamps to provide a scalable, tamper-resistant integrity substrate. The design prioritizes low on-chain footprint and off-chain data retention, enabling high-frequency commitments without congesting the network.

Limitations: PoCI does not verify the *content* of computations, nor does it guarantee correctness of model logic or training data. It relies on honest event generation, secure key management, and external storage for full artifacts. It does not replace formal verification or ZK-based compute proofs; instead, it complements them by securing *when*, *by whom*, and *in what sequence* AI computations occurred.

PoCI aims to establish a pragmatic, scalable foundation for AI accountability - bridging cryptographic auditability, distributed consensus, and real-world model behavior in a way existing systems do not.

1. Introduction

Artificial intelligence systems now generate text, decisions, diagnoses, financial signals, and policy-relevant outputs at scales that profoundly impact society. Yet there is still **no reliable way to verify that an AI model actually performed the computation it claims**, nor when, nor under what internal state. Existing approaches to AI accountability depend on mutable logs, centralized API telemetry, unverifiable metadata, or proprietary audit trails that can be forged or retroactively altered.

This lack of compute integrity creates systemic vulnerabilities:

- **Outputs can be fabricated** without performing any computation.
- **Timestamps can be backdated** to justify false claims or hide misconduct.
- **Lineage can be rewritten**, breaking scientific reproducibility and legal traceability.
- **Models can drift** or be tampered with while maintaining a false appearance of continuity.
- **Regulators and auditors have no cryptographic ground truth** to verify claims.

The problem is not the absence of data - it is the absence of **trustworthy structure**.
AI systems operate with no cryptographic binding between *what* was computed, *when* it occurred, and *which model version* performed it.

1.1 Limitations of Existing Approaches

Current provenance or “AI audit” solutions suffer from significant weaknesses:

- **Centralized logs** can be edited or deleted.
- **API telemetry** relies on the honesty of the provider.
- **Model registries** record versions but not *compute events*.
- **ZKML and verifiable inference** prove correctness, not *authorship or timing*.
- **Blockchain integrations** typically store large data or rely on low-throughput L1s.

Critically, most solutions assume static agent behavior (game-theoretic stability), ignoring how real systems and adversaries *adapt over time*.

1.2 Why Anchoring Compute to a PoW BlockDAG Matters

For AI provenance to be trustworthy, compute events must be:

1. **Cryptographically signed** by the model or agent executing them.
2. **Timestamp-anchored** to a ledger that cannot be rewritten.
3. **Ordered** to reconstruct lineage and model evolution.

4. **Immutable** under adversarial conditions.
5. **Scalable enough** to handle high-frequency compute commitments.
   Kaspa's **Proof-of-Work blockDAG** uniquely satisfies these constraints:

- Parallel block processing enables **high throughput**.
- The DAG structure yields **predictable, observable finality**.
- PoW provides **physics-anchored timestamps** that resist manipulation.
- Merge-depth rules prevent **retroactive ordering attacks**.
- Low-latency block production supports **frequent commitments**.
  This combination makes Kaspa not just a blockchain, but a *timestamp integrity substrate* suited for verifiable AI compute.

1.3 Contribution of This Work

This paper introduces **Proof of Compute Integrity (PoCI)** - a practical, low-bloat framework for transforming AI compute events into verifiable, privacy-preserving commitments anchored to Kaspa.

PoCI provides:

- A **four-loop architecture** for event creation, proof generation, on-chain anchoring, and lineage verification.
- An **adaptive integrity controller** that adjusts verification strictness based on model behavior, aligning with learnability-equilibrium principles.
- A separation of **off-chain artifacts** and **on-chain commitments**, minimizing blockchain storage needs.
- A path toward **scalable, decentralized AI accountability** without exposing sensitive model details or requiring protocol-level changes to the AI system itself.
-

1.4 Scope and Limitations

PoCI does **not** verify the correctness of AI outputs or the internal reasoning of models.
It does not guarantee dataset integrity or replace full ZK-compute proofs.
Instead, PoCI provides a missing primitive: **cryptographically verifiable evidence of when, by whom, and in what sequence compute took place.**

This makes PoCI compatible with - and complementary to - future advances in ZKML, hardware attestation, and formal verification.

2. Background & Motivation

We detail the motivation for Proof of Compute Integrity by analyzing the gaps in current AI provenance approaches and showing why a high-throughput PoW blockDAG like Kaspa is uniquely positioned to address them.

2.1 The Structural Problem: AI Without Integrity

As artificial intelligence systems become integrated into financial markets, healthcare diagnostics, scientific research, autonomous decision-making, and national-security operations, the absence of **cryptographic guarantees around computation** has become a foundational weakness.

Today, AI systems operate within an integrity vacuum:

- **No proof of computation:**
  A model can claim to have run an inference or analysis without ever performing it.
- **No binding timestamps:**
  Outputs can be generated now and claimed to have occurred earlier (or later), undermining auditability.
- **No lineage continuity:**
  Model versions, checkpoints, and updates can be altered without a verifiable historical trace.
- **No tamper resistance:**
  Internal logs, model registries, and telemetry streams are easy to forge or delete.
- **No decentralized verification:**
  Auditors must trust centralized infrastructure controlled by the model provider.
  The result is a world where **AI claims are unverifiable by default** - a dangerous condition as models are entrusted with increasingly consequential tasks.

2.2 Why Traditional Provenance and Logging Approaches Fail

A variety of provenance systems exist, but none solve the integrity problem at a cryptographic level.

Centralized Logs

Server logs, API telemetry, or proprietary audit trails rely entirely on the honesty of the operator. They can be edited, overwritten, or selectively disclosed.

Model Registries

Tools that track model versions (weights, checkpoints, configurations) do not verify *compute events* . They record what a model *is*, not what it *did*.

Secure Enclaves & Hardware Attestation

These prove computations occurred inside trusted hardware, but:

- they require specialized hardware,
- can be compromised by firmware exploits,
- and do not provide decentralized verifiability.
-
- ## Zero-Knowledge Machine Learning (ZKML)
  ZKML can prove *correctness* of specific operations, but:
- does not prove **authorship**,
- does not bind compute to **wall-clock time**,
- is computationally expensive for high-frequency inference,
- and still requires an external anchor for commitment ordering.
-
- ## Blockchain-Based Approaches Today
  Most existing “AI + blockchain” systems:
- rely on low-throughput L1s not suited for high-frequency compute events,
- store too much data directly on-chain,
- or attempt to build entire AI-execution layers rather than a verifiable integrity substrate.
  None provides the **timing guarantees** or **lineage ordering** required for real-world auditability.##

2.3 Why Timestamp Integrity is the Missing Primitive

In AI systems, *when* computation occurred is often as important as *what* was computed.
A model output is trustworthy only if:

- it was produced by the claimed model version,
- at the claimed time,
- in the claimed sequence relative to other events.
  Without timestamp integrity:
- backdating becomes trivial,
- tampering goes undetected,
- model evolution cannot be verified,
- and legal/forensic analysis becomes impossible.
  This is not a problem ZK, logging, or hardware alone can solve.
  It requires an external **immutable ordering substrate**.

2.4 Why an L1 Blockchain Anchor is Necessary - But Not Sufficient

A public blockchain provides:

- immutability,
- decentralized ordering,
- timestamp anchoring,
- and auditability without trusting a single party.
  However, not all chains are suitable.

PoS chains introduce governance-driven reordering risk, MEV concerns, and stake-based control that weakens integrity.

Low-throughput chains cannot anchor thousands of daily compute events.

High-fee ecosystems make continuous anchoring economically infeasible.

Chains without predictable finality cannot provide reliable event ordering.

Thus, an L1 is necessary - but only a *specific type* of L1 can support real compute integrity.

2.5 Why Kaspa's PoW BlockDAG is Uniquely Suitable

Kaspa’s architecture provides properties **not found together anywhere else**:

Parallel Block Processing (BlockDAG)

Allows anchoring of high-frequency compute events without congestion.

Predictable, physics-anchored finality

PoW + DAG structure + merge-depth rules give:

- stable finality,
- timestamp integrity,
- and resistance to reordering attacks.
- **Low latency (approx. 10 seconds)**
  Supports near-real-time anchoring.

No premine, no insiders, no governance capture

Ensures the integrity substrate is not politically or economically controlled by any AI provider.

Future Vprog support

Allows programmable verification logic and zero-knowledge extensions as the system matures.
Kaspa is not just fast - it is structurally aligned with the **requirements of AI compute integrity**.

2.6 Motivation for PoCI

PoCI emerges from the recognition that:

- AI requires external, cryptographic verification,
- blockchains provide immutable ordering,
- but existing designs do not connect compute → proof → timestamp in a scalable way.

PoCI provides:

- a **lightweight, four-loop architecture** for transforming compute events into immutable commitments,
- an **adaptive integrity controller** for shaping model behavior,
- and a clean separation between **off-chain artifacts** and **on-chain proofs** to minimize bloat.

Its motivation is simple:

To create a practical, verifiable, decentralized integrity layer for AI systems - something that has been missing until now.

3. System Overview

3.1 Architectural Goals

Proof of Compute Integrity (PoCI) is designed to provide a practical, scalable, and cryptographically verifiable method for anchoring AI compute events to an immutable ledger.
The architecture is guided by four goals:

1. **Integrity:**
   Ensure compute events cannot be backdated, altered, or fabricated.
2. **Lineage:**
   Allow reconstruction of a model's evolution and sequence of operations across its lifetime.
3. **Scalability:**
   Support high-frequency commitments without congesting the underlying chain.
4. **Privacy:**
   Provide verifiable proofs without exposing sensitive model weights, proprietary logic, or input data.
   These goals shape the structure of PoCI and dictate the separation between off-chain artifacts and on-chain commitments.

3.2 Core Components

PoCI is built around five core components that work together to provide a full compute-integrity pipeline:

(1) Compute Events

AI models emit structured “events” each time they perform a meaningful operation (e.g., inference, analysis, update, checkpoint).
These events include:

- model ID
- event ID
- timestamp
- hashed metadata
- optional descriptors
- signature
  They form the foundational unit of lineage.

(2) Proof Artifacts

Each event produces a lightweight cryptographic proof:

- A commitment hash to the event data
- A model-authorship signature
- Optional partial-reveal or zero-knowledge attestations
  These proofs are privacy-preserving and separated from the full event payload.

(3) On-Chain Commitments

Only the compact proof (a hash + signature bundle) is committed to the Kaspa blockDAG.
This provides:

- immutable ordering
- decentralized timestamping
- tamper resistance
- verifiable provenance
  Kaspa’s DAG parallelism allows thousands of such commitments per day without congestion.

(4) Lineage Graph

Clients, auditors, or verifiers can retrieve event artifacts from off-chain storage and reconstruct a model's:

- evolution
- state transitions
- behavioral history
- compute chronology
  This forms a cryptographically backed lineage graph tied to immutable blockchain timestamps.

(5) Adaptive Integrity Controller

A supervisory mechanism that:

- detects anomalies,
- increases verification pressure on suspicious models,
- adjusts proof strictness,
- and maintains long-term system honesty.
  This shifts PoCI from static game theory toward a dynamic learnability equilibrium.

3.3 High Level Data Flow

The PoCI system operates as a pipeline:

1. **Model performs compute** → emits event
2. **Event is hashed + signed** → produces proof
3. **Proof is anchored on Kaspa** → timestamp, ordering, immutability
4. **Auditors retrieve events** → verify signatures + lineage
5. **Adaptive controller evaluates behavior** → adjusts verification requirements
   This flow ensures that every compute action has:

- a verifiable origin,
- a tamper-resistant timestamp,
- an immutable position in model history,
- and a proof that the event actually occurred.

**3.4 Separation of Concerns

PoCI deliberately separates:

Off-Chain Data

- Full event payload

- Optional model metadata
- Zero-knowledge artifacts
- Checkpoints / logs
- Input classifications
- Storage systems (IPFS, Filecoin, enterprise storage)
- **On-Chain Data**
- A small hash commitment
- A signature
- Minimal metadata
  This minimizes on-chain bloat while preserving maximum verifiability.

3.5 Compatibility and Extensibility

PoCI is designed to be:

Chain-Agnostic (Conceptually)

While optimized for Kaspa, the framework can anchor to any sufficiently:

- high-throughput,
- tamper-resistant,
- decentralized,
- finality-stable ledger.
- **Model-Agnostic**
  Works with:
- LLMs
- diffusion models
- agentic pipelines
- predictive models
- reinforcement-learning systems
- **ZK-Compatible**
  As ZKML matures, PoCI serves as a timestamp and lineage substrate rather than a correctness oracle.

3.6 Summary

The PoCI system provides a structured, scalable architecture for verifying *when*, *by whom*, and *in what sequence* AI computations were performed - without storing sensitive information on-chain or requiring changes to existing AI frameworks.

It transforms compute from an unverifiable black box into a cryptographically accountable pipeline.

4. Four-Loop Compute Integrity Architecture

4.1 Event Creation

The first loop defines how an AI system transforms an internal computation into a structured, externally verifiable event. This loop establishes the *ground truth* from which all downstream proofs and lineage reconstruction are derived. Its design ensures that every meaningful computation produces a consistent, signed artifact that can later be verified against its on-chain commitment.

4.1.1 Purpose

Loop 1 ensures that each AI computation:

- produces a standardized, machine-readable record,
- is bound to the identity of the executing model or agent,
- cannot be retroactively modified,
- and contains sufficient metadata for future lineage reconstruction.
  This creates a canonical representation of “what the model claims it did” before any hashing, proof generation, or on-chain anchoring occurs.

4.1.2 Event Structure

Each compute event is a structured object that captures the minimum necessary information for verifiable provenance without exposing sensitive model internals.
A typical event includes:

- **model_id:** cryptographic identifier of the model
- **event_id:** unique nonce or sequential ID
- **timestamp_local:** the model’s local timestamp
- **weights_hash:** hash of model weights or parameters (optional or partial)
- **input_class:** non-sensitive classification of the input (category, domain, type)
- **output_hash:** optional hash of the model output (not the output itself)
- **prev_event_id:** pointer linking lineage
- **metadata:** optional descriptors (task type, context, workload)
- **signature:** model or agent signature binding the event to its identity
  The complete schema will be formally defined in Section 6 (Specifications).

4.1.3 Signing and Authorship Binding

Before leaving the model’s local environment, the event is **signed** using the model’s private key.
This provides:

- **authorship binding** - proves which model produced the event
- **tamper resistance** - any modification invalidates the signature
- **non-repudiation** - models cannot deny generating the event
  Unlike centralized logging solutions, authorship is encoded cryptographically, not socially or administratively.

4.1.4 Local Timestamping (Pre-Anchor)

PoCI deliberately records two timestamps:

1. **Local timestamp** (inside the event)
2. **Chain-anchored timestamp** (added in Loop 3)

The local timestamp is *not* trusted on its own.
Its role is to:

- help reconstruct local ordering,
- support anomaly detection in the adaptive controller,
- and allow comparison against the immutable Kaspa timestamp in Loop 3.
  Any significant deviation between local and on-chain time acts as a dishonesty signal.

4.1.5 Event Minimization and Privacy

Loop 1 enforces a strict principle:
Record only what is required for verifiable lineage - nothing more.
This prevents:

- leakage of proprietary model weights,
- exposure of sensitive input data,
- revealing full outputs,
- storage of internal activations,
- or unnecessary details that increase attack surface.
  PoCI is not a logging system; it is an **integrity system**.

4.1.6 Linkage and Chain of Custody

Every event references its predecessor (`prev_event_id`) to form an ordered, tamper-resistant chain of custody before anchoring.
This acts as:

- an internal lineage map,
- a pre-hash commitment chain,
- and a consistency check for later verification.
  Even before anchoring, the model cannot rewrite prior events without breaking the local chain.
  Once Loop 3 anchoring occurs, rewriting becomes *impossible* without detection.

4.1.7 Summary of Loop 1

Loop 1 converts raw AI computation into a structured, signed, privacy-safe, lineage-aware event.
It does **not** attempt verification or anchoring - it simply prepares the canonical record that all later loops depend on.
Key guarantees provided:

- consistent event representation
- cryptographically bound authorship
- privacy-preserving metadata
- pre-lineage continuity
- readiness for proof generation
  Loop 1 is the foundation of the PoCI pipeline; the next loop transforms these events into cryptographic proofs.

4.2 Proof Generation

Loop 2 transforms a signed compute event into a tamper-resistant cryptographic proof that can be anchored to the Kaspa blockDAG. This loop formalizes the event into a commitment that proves **authorship, integrity, and continuity**, while preserving the privacy of model internals and input/output data.
Loop 2 is where PoCI shifts from *structured logging* to *cryptographic evidence*.

4.2.1 Purpose

Loop 2 exists to answer three questions with cryptographic certainty:

1. **Did this exact event exist?**
   → Guaranteed by hashing and signing.
2. **Did the claimed model create it?**
   → Guaranteed by authorship binding via Ed25519 signatures.
3. **Can this event be linked to its predecessor and future events?**
   → Guaranteed by the lineage hash chain.

This loop creates the minimal cryptographic artifact required to later anchor events on-chain.

4.2.2 Commitment Hash Construction

The event object from Loop 1 is serialized into a canonical format (e.g., JSON with deterministic key ordering). It is then hashed using a collision-resistant hash function such as **SHA-256** or **BLAKE3**.

commit_hash = H(event_data)

This hash commits to:

- event metadata
- local timestamp
- weights hash (full or partial)
- input classification
- output hash (if included)
- `prev_event_id`
- and all other fields of the schema
  The commitment hash ensures that **any alteration** to any field of the event produces a different hash, making tampering detectable.

4.2.3 Authorship Signature

To bind the commit hash to the model, the model signs the hash with its private key:

signature = Sign(model_private_key, commit_hash)

This signature provides:

- **authorship proof** - only the model with the correct private key could produce it
- **tamper detection** - editing the event invalidates the signature
- **non-repudiation** - the model cannot deny generating the event later
  The corresponding public key forms part of the model's identity (`model_id`).

4.2.4 Lineage Hash Chain

To ensure sequential integrity of events, Loop 2 incorporates the previous event's hash into the commitment:

lineage_hash = H(commit_hash || prev_commit_hash)

This creates a **hash-linked chain of events**, similar to a blockchain—but local to the model.
Benefits:

- Prevents reordering of events
- Prevents deletion of past events
- Prevents insertion of fabricated events mid-history
- Strengthens the detectability of manipulated lineage
  Later loops (Loop 3 and Loop 4) verify that the external (on-chain) ordering matches the internal lineage chain.

4.2.5 Optional Zero-Knowledge or Partial-Reveal Extensions

PoCI is designed to be privacy-preserving.
For models handling sensitive data, Loop 2 supports optional enhancements:

(1) Partial Reveals

Reveal only:

- hashes of outputs
- semantic class of inputs
- task descriptors
  Useful for enterprise or classified systems.

(2) Zero-Knowledge Attestations

Future-proof support for:

- selective proof of model properties
- selective proof of input classifications
- binding event metadata without exposing it
- ZK inference correctness proofs (optional, not required)
  These are not required for the minimal PoCI prototype but provide extensibility for high-security industries.

4.2.6 Data Minimization Principles

Loop 2 follows three strict constraints:

1. **No full outputs on-chain**
2. **No model weights on-chain**
3. **No sensitive data revealed**

All verifiability is derived from:

- hashes,
- signatures,
- ordering guarantees,
- and timestamp anchoring.

This ensures PoCI remains safe for:

- proprietary models
- regulated industries
- government systems
- and commercial deployments
  without risking intellectual property leakage.

4.2.7 Proof Object Format

The final proof object produced by Loop 2 (to be anchored in Loop 3) consists of:

```
{
"model_id": "...",
"commit_hash": "...",
"lineage_hash": "...",
"signature": "...",
"prev_commit_hash": "..."
}
```

This is compact enough to be included in a transaction payload on Kaspa with minimal bloat.

4.2.8 Summary of Loop 2

Loop 2 transforms a raw event (Loop 1) into a **cryptographic proof**.
It creates a **tamper-evident, signed, lineage-linked commitment** that can be anchored to Kaspa for immutable timestamping.

Guarantees provided:

- Event integrity
- Authorship authenticity
- Lineage continuity
- Privacy preservation
- Readiness for on-chain anchoring
  Loop 3 will now bind that proof to physical time and immutable ordering.

4.3 On-Chain Anchoring

Loop 3 anchors the cryptographic proof produced in Loop 2 to the Kaspa PoW blockDAG.
This is where PoCI transitions from **local integrity** to **global immutability**.
The anchoring step binds compute events to:

- physical time,
- decentralized ordering,
- and tamper-proof finality.
  Anchoring is what prevents a model from rewriting history, backdating events, or creating fraudulent lineage.

4.3.1 Purpose

Loop 3 ensures three properties that cannot be achieved without an external consensus layer:

(1) Immutable Timestamping

Kaspa's PoW mechanism produces timestamps tied to:

- real-world energy expenditure,
- block-generation physics,
- and DAG-based ordering rules.
  Once anchored, an event's timestamp cannot be altered.

(2) Immutable Ordering

The DAG structure orders proofs relative to each other.
This prevents:

- reordering events to hide tampering,
- inserting fabricated "older" events,
- manipulating lineage for malicious purposes.

(3) Decentralized Attestation

No single model provider controls the ledger.
Anchoring makes compute provenance **trustless**.

4.3.2 Kaspa as the Integrity Substrate

Kaspa's architecture provides properties essential for AI compute anchoring:

• BlockDAG Parallelism

Multiple blocks per second allow **continuous, high-frequency commits**.

• Predictable Finality via Merge-Depth

After a fixed number of DAG layers, reordering becomes mathematically infeasible.

• PoW-Based Timestamps

Unlike PoS, timestamps cannot be manipulated by governance or stake-weight collusion.

• Low Latency (~10 seconds)

Allows near-real-time anchoring of compute events.

• Fair Launch and Decentralization

No private actors can rewrite history or selectively censor AI commitments.
These properties are why PoCI anchors to Kaspa instead of a generic L1.

4.3.3 Anchoring Transaction Structure

The lightweight proof object from Loop 2 is included in a Kaspa transaction:

```
{
"model_id": "...",
"commit_hash": "...",
"lineage_hash": "...",
"signature": "..."
}
```

Nothing sensitive is included.

This commitment is written into Kaspa's DAG as transaction metadata.

- The **commit_hash** proves exact event integrity
- The **signature** binds it to the model
- The **lineage_hash** ensures continuity
- Kaspa provides the **timestamp** and **ordering**
  The transaction is then propagated, validated, and mined via PoW consensus.

4.3.4 Timestamp Anchoring

The timestamp on a PoW DAG is not arbitrary - it is derived from:

- block acceptance rules,
- network propagation windows,
- median-time adjustments,

- and hash-rate constraints.
  A model cannot:
- forge an earlier timestamp,
- anchor an event "in the past,"
- or bypass the merge-depth finality window.
  This single property alone breaks the possibility of backdated or fabricated model history.

4.3.5 Ordering Guarantees via DAG Structure

Kaspa's DAG orders blocks in partial order.
This is critical for lineage integrity:

- Two events committed at time T0 and T1 cannot be swapped retroactively.
- Events anchored after a checkpoint cannot be inserted before it.
- Fraudulent "historic" events fail because merge-depth enforces temporal constraints.
  This ensures that lineage reconstructed in Loop 4 matches reality - not a rewritten version.

4.3.6 Cost and Scalability Considerations

PoCI minimizes on-chain footprint by anchoring **one hash + one signature per event**.
Kaspa's scalability supports:

- thousands of commits per day, per model cluster,
- multiple organizations anchoring in parallel,
- high-frequency auditing without chain congestion.
  Fees remain negligible due to:
- high throughput,
- efficient fee market,
- compact transactions.
  This avoids the L1 bloat problem plaguing many blockchain-based provenance systems.

4.3.7 Security Guarantees from Anchoring

Loop 3 provides the following security properties:

Against backdating

Impossible due to PoW timestamp integrity.

Against lineage rewriting

Impossible due to DAG ordering and merge-depth immutability.

Against selective omission

Detectable because lineage_hash continuity breaks.

Against forging events

Detection happens because signatures will not match commit_hash.

Against collusion

PoW removes governance-based manipulation vectors.
Anchoring is the single most important step that converts Loop 1–2 data into trustless evidence.

4.3.8 Summary of Loop 3

Loop 3 binds compute events to:

- **time**,
- **order**,
- **immutability**,
- **and decentralized truth.**
  It converts cryptographic proofs into **global, verifiable integrity primitives**.
  This is what allows Loop 4 to reliably reconstruct model history.

4.4 Loop 4 - Verification and Lineage Reconstruction

Loop 4 is the final stage of the PoCI pipeline. It enables external parties - auditors, regulators, researchers, or automated systems - to **reconstruct a model's entire compute history**, verify the authenticity of events, detect tampering, and evaluate model behavior over time.

Loop 4 is where the promises of Loops 1–3 become **observable, testable reality.**

4.4.1 Purpose

Loop 4 answers the ultimate question:
Did the model actually perform these computations, at these times, in this order — and can this story withstand adversarial scrutiny?
Verification includes:

- validating signatures
- matching hashes
- checking lineage continuity
- comparing timestamps
- inspecting anchoring consistency

- identifying tampering or omissions
  This loop transforms the anchored commitments into a **cryptographically verifiable lineage graph.**

4.4.2 Inputs to Loop 4

A verifier receives three streams of data:

(1) On-chain commitments

From Loop 3:

- commit_hash
- lineage_hash
- signature
- anchor timestamp
- transaction ID

(2) Off-chain full event data

From Loop 1:

- metadata
- input class
- weights hash
- output hash
- previous event pointer
- local timestamp

(3) Model identity and public key

Used to validate signatures.
All three combined provide everything needed for full verification.

4.4.3 Signature Verification

The verifier checks:
VerifySignature(model_public_key, commit_hash, signature) == TRUE
If this fails:

- the event is forged, OR
- the model key has been compromised, OR

- the event data was altered after signing.
  This is the first line of defense.

4.4.4 Hash Integrity Validation

The verifier recomputes:
H(event_data) == commit_hash
If they differ:

- the event data was tampered, OR
- the provider is attempting manipulation.
  This ensures **event integrity**.

4.4.5 Lineage Reconstruction

Using `prev_event_id` and `prev_commit_hash`, the verifier reconstructs the model's event chain:
Event_1 → Event_2 → Event_3 → ... → Event_N
The verifier checks:

- the lineage is continuous
- no missing links
- no duplicated events
- no forked histories
- no retroactive rewriting
  Equivalent to reconstructing a local blockchain created by the model.
  This chain is then compared against the **on-chain ordering** provided by Kaspa.
  If the sequences diverge, tampering has occurred.

4.4.6 Timestamp Cross-Validation

Loop 4 compares:

Local timestamp (from Loop 1)

vs

On-chain anchor timestamp (from Loop 3)

Acceptable drift exists (network delays, batching, etc.), but significant discrepancies indicate:

- delayed anchoring
- hidden events
- fabricated history
- model clock manipulation

- anchoring only “clean” events while hiding others
  This cross-check is one of PoCI’s strongest integrity tools.

4.4.7 Detecting Omission, Reordering, and Fabrication

Loop 4 detects three classes of manipulation:

(1) Event Omission

Missing commit_hashes or gaps in lineage reveal withheld computations.

(2) Event Reordering

If chronological order conflicts with on-chain anchor order → manipulation.

(3) Fabricated Events

Events with:

- invalid signatures
- mismatched hashes
- lineage inconsistencies
- abnormal timestamps

…are rejected as fraudulent.
Loop 4 is adversarial by design - it assumes actors may lie unless proven otherwise.

4.4.8 Lineage Graph Construction

After verification, the verifier builds a **lineage graph**:
Nodes = compute events
Edges = parent → child relationships
This graph enables:

- full audit trails
- model-behavior analysis
- anomaly detection
- compliance reporting
- scientific reproducibility
- legal traceability
  Because of on-chain anchoring, this graph is tamper-resistant and globally verifiable.

4.4.9 Interoperability with Zero-Knowledge Systems

Loop 4 does not replace ZK proofs.
Instead, it complements them by:

- providing immutable timestamps,
- verifying authorship,
- establishing event sequence,
- and linking all ZK-verified operations into a coherent compute history.
  In future versions, Loop 4 may incorporate:
- ZK attestations for input classification
- ZK commitments for model version integrity
- ZK partial proofs for sensitive computations
  These plug directly into PoCI's verification pipeline.

4.4.10 Summary of Loop 4

Loop 4 validates:

- **authorship**
- **integrity**
- **ordering**
- **timestamp accuracy**
- **lineage continuity**
- **event authenticity**
  It reconstructs the model's compute lifecycle in a way that **cannot be forged, rewritten, or selectively deleted**, due to the combination of:
- local signatures (Loop 2)
- hash commitments (Loop 2)
- DAG timestamps (Loop 3)
- lineage pointers (Loop 1)
  Loop 4 is the final stage that makes PoCI a complete, verifiable AI integrity framework.

5. Adaptive Integrity Controller

While the Four-Loop architecture provides a complete pipeline for generating, anchoring, and verifying compute events, it alone cannot guarantee *long-term behavioral honesty*.
AI systems are not static; they adapt, optimize, and sometimes exploit blind spots in their verification process.

The Adaptive Integrity Controller (AIC) is PoCI's meta-layer that introduces **dynamic trust-pressure**, enabling the system to respond in real time to anomalous behavior. Instead of relying on fixed assumptions about honest agents, the AIC aligns PoCI with **learnability equilibrium** -

a framework in which adversaries adapt, verifiers adapt in response, and the system stabilizes through continuous feedback.

The AIC is **not** part of the compute pipeline.
It **observes** the loops.
It **evaluates** behavior.
It **adjusts** verification requirements.
It **hardens the system over time**.

5.1 Purpose

The AIC exists to solve problems that static verification systems inherently fail to address:

1. **Models learn to exploit fixed rules.**
   Static checks become predictable and thus vulnerable.
2. **Tampering becomes subtle over long timelines.**
   Gradual drift is difficult to detect without dynamic baselines.
3. **High-frequency compute systems behave non-linearly.**
   Sudden spikes or drops in activity may indicate hidden behavior.
4. **Different models require different levels of scrutiny.**
   A one-size-fits-all verification policy is inefficient and unsafe.
   The AIC makes PoCI adaptive, contextual, and behavior-aware.

5.2 Observability Across the Four Loops

The Adaptive Integrity Controller monitors **behavioral signals** produced by each loop.

Signals from Loop 1 - Event Creation

- irregular intervals between events
- missing or suppressed events
- inconsistent input-class or metadata patterns
- unusual changes in weights_hash
- unexpected lineage-pointer jumps

Signals from Loop 2 - Proof Generation

- repeated commit_hashes (possible replay)
- malformed proofs
- invalid signatures
- lineage hash inconsistencies
- drop-offs in proof frequency

Signals from Loop 3 - On-Chain Anchoring

- large delays between event creation and anchoring
- mismatched local vs blockchain timestamps
- clustering of events to hide irregular history
- gaps in anchoring sequence
- abnormal anchoring costs (possible batching attacks)

Signals from Loop 4 - Verification

- broken lineage
- missing off-chain artifacts
- impossible event sequences
- timestamp manipulation attempts
- model behaviors inconsistent with historical baselines
  The combination of all these signals forms the model's **real-time integrity profile**.

5.3 Adaptive Response Mechanisms

When anomalies are detected, the AIC adjusts verification strictness through several mechanisms:

(1) Increased Proof Requirements

- more frequent commitments
- inclusion of output hashes
- mandatory lineage-chain strengthening
- selective zero-knowledge attestations

(2) Tightened Timestamp Tolerance

- reduced acceptable deviation between local and on-chain timestamps
- stricter rejection of delayed anchoring
- event patterns examined under higher scrutiny

(3) Trust Score Adjustment

Each model maintains a dynamic trust score based on:

- consistency
- reliability
- anomaly rate

- historical behavior
  Lower trust → higher scrutiny.

(4) Anomaly-Triggered Full Audits

Significant irregularities trigger:

- full lineage reconstruction
- multi-loop cross-verification
- historical drift analysis
- signature and chain-consistency checks

(5) Automated Behavior Shaping

The AIC gradually shapes model behavior by:

- reducing the value of incomplete proofs
- penalizing suspicious anchoring behavior
- elevating the detail required in event metadata
- increasing anchoring frequency under suspicion
  This turns PoCI into a **feedback-driven integrity system**.

5.4 Learnability Equilibrium vs Static Game Theory

Traditional systems rely on **game-theoretic stability**, assuming static adversaries acting under known incentives.
But AI systems and human operators:

- adapt
- optimize
- probe the system
- exploit predictable rules
- learn to evade detection
  PoCI aligns with **learnability equilibrium**, where:
- verifiers adapt
- models adapt
- equilibrium emerges through mutual adjustment
  The Adaptive Integrity Controller is the mechanism that operationalizes this principle.

5.5 Deployment Models

The Adaptive Integrity Controller can be deployed in multiple configurations:

A. Client-Side / Auditor-Side (recommended)

Runs within external audit software or verifiers.
Pros:

- decentralized
- no reliance on model operator
- scalable across organizations

B. Model-Side Agent

Runs beside the model itself as a local monitoring daemon.
Pros:

- real-time anomaly detection
- immediate self-regulation
  Cons:
- requires cryptographic attestation to avoid self-reporting loopholes

C. Distributed Integrity Network (future extension)

Multiple verifiers maintain a shared trust index.
Pros:

- strongest security
- ecosystem-wide monitoring
  Cons:
- requires broad ecosystem adoption

5.6 Integrity Signals and Threat Detection

The AIC uses multivariate signals to detect:

- forged events
- timestamp manipulation
- lineage erasure
- delayed anchoring
- suppressed compute
- key compromise
- behavioral drift
- computational evasion strategies
  These signals feed the trust-pressure equations in the next section.

5.7 Formal Integrity-Pressure Dynamics (12g-ACE and 12g-AFL)

Two original adaptive-update functions formalize PoCI’s trust-feedback system:

12g-ACE: Adaptive Compute Expectation

The expected verification pressure at time $t+1t+1t+1$:
$Pt+1=Pt+\alpha\cdot At-\beta\cdot Ct$
Where:

- Pt = current verification pressure
- At = anomaly score
- Ct = consistency score
- α = anomaly sensitivity coefficient
- β = stability reinforcement coefficient

Meaning:

If anomalies rise → pressure increases.
If the model behaves consistently → pressure relaxes.
The verifier “learns” how much scrutiny each model requires.

12g-AFL: Adaptive Forgery Likelihood

The estimated probability that a model is attempting dishonest or evasive behavior:
$Ft+1=\gamma\cdot Ft+\delta\cdot At+\epsilon\cdot Dt$
Where:

- Ft = current forgery likelihood
- At = anomaly score
- Dt = timestamp drift
- γ = temporal decay
- δ = anomaly weighting
- ϵ = drift sensitivity

Meaning:

Repeated anomalies compound.
Timestamp drift increases suspicion.
Older behavior decays over time, enabling recovery.
This produces a probabilistic dishonesty estimator.

5.8 Summary

The Adaptive Integrity Controller converts PoCI from a static audit pipeline into a **dynamic, behavior-aware integrity system**. It continuously evaluates model behavior, adjusts verification strictness, and enforces a form of learnability equilibrium that discourages evasion and encourages long-term honesty.
It is this adaptive mechanism - encapsulated in the 12g-ACE and 12g-AFL update rules - that makes PoCI a living system, capable of scaling alongside evolving AI models and adversarial strategies.

6. Formal Specifications

This section defines the formal data structures, types, and rules that constitute the Proof of Compute Integrity (PoCI) protocol. These specifications provide a reference implementation for developers, auditors, and integrators. While PoCI is conceptually flexible, the structures defined here represent the canonical v1.0 specification.
PoCI relies on minimal, composable components:

- a standardized **event schema**,
- a deterministic **proof object**,
- a compact **anchoring payload**, and
- a well-defined **verification procedure**.
  These specifications ensure interoperability between models, verifiers, and auditors across different organizations and environments.

6.1 Event Schema (Loop 1)

An event is a structured representation of a single meaningful computation performed by an AI model or agent.
All fields must be **deterministically serialized** (e.g., lexicographically ordered JSON keys) before hashing.

6.1.1 Event Object

```
Event {
string model_id
string event_id
string prev_event_id
uint64 timestamp_local
bytes32 weights_hash // optional or partial
string input_class // semantic category only
bytes32 output_hash // optional
map<string, string> metadata // optional descriptors
}
```

Field Requirements;

Field	Purpose	Required
`model_id`	Public key hash identifying the model	Yes
`event_id`	Unique nonce or sequence for local ordering	Yes
`prev_event_id`	Forms local lineage chain	Yes
`timestamp_local`	Local timestamp at event creation	Yes
`weights_hash`	Optional commitment to model state	Optional
`input_class`	Non-sensitive input category	Recommended
`output_hash`	Hash of output (not raw output)	Optional
`metadata`	Optional descriptive info	Optional

All optional fields that are omitted must be explicitly encoded as `null` to maintain deterministic hashing.

6.2 Proof Object (Loop 2)

A proof object is a minimal cryptographic commitment to an event.

6.2.1 Commitment Hash

$commit\_hash = H(Event)$

6.2.2 Lineage Hash

$lineage\_hash = H(commit\_hash \| prev\_commit\_hash)$
Where:

- `prev_commit_hash` is the committed hash of the previous event.
- For a genesis event, `prev_commit_hash = 0x00…00`.

6.2.3 Signing Rule

The model signs its commit hash using Ed25519:
$signature = Sign(skmodel, commit\_hash)$
Where `sk_model` is the model's private key.

6.2.4 Proof Object Structure

Proof {
bytes32 commit_hash

```
bytes32 lineage_hash
string model_id
bytes signature
bytes32 prev_commit_hash
}
```
This object is the payload that will be anchored to Kaspa in Loop 3.

6.3 Anchoring Payload (Loop 3)

PoCI uses extremely compact anchoring transactions to avoid chain bloat.

6.3.1 Kaspa Payload Serialization

```
KaspaCommit {
bytes32 commit_hash
bytes32 lineage_hash
string model_id
bytes signature
}
```
Only the minimal integrity-critical fields are included on-chain.
The full event remains off-chain.

6.3.2 Transaction Requirements

The payload must be included in a Kaspa transaction using:

- OP_RETURN-like metadata (in Kaspa's data field), or
- a dedicated PoCI commitment field (future extension), or
- a multi-field data script (recommended for compactness).
  Kaspa's DAG finalizes ordering after the protocol-defined merge-depth.

6.4 Verification Rules (Loop 4)

A verifier reconstructs lineage and authenticates events using the following rules.

6.4.1 Rule 1 - Signature Validity

$\mathrm{Verify}(pkmodel, commit\_hash, signature) = TRUE$
If false → discard event as invalid.

6.4.2 Rule 2 - Hash Integrity

Recompute:
$H(Event) = commit\_hash$
If mismatch → event tampering.

6.4.3 Rule 3 - Lineage Continuity

Recompute:
$H(\text{commit\_hash} \| \text{prev\_commit\_hash}) = \text{lineage\_hash}$
Compare against:

- the previous event's commit hash
- the on-chain sequence
  If lineage breaks → manipulation detected.

6.4.4 Rule 4 - Timestamp Cross-Validation

Let:

- Tlocal == timestamp_local from event
- Tchain = block timestamp from Kaspa anchoring
  Compute drift:
  $D = |T_{local} - T_{chain}|$
  If D exceeds tolerance threshold → flag anomaly for AIC.

6.4.5 Rule 5 - Ordering Consistency

A verified event must satisfy:

- on-chain order ≈ local lineage order
- no event anchored after its descendant
- no event inserted between earlier ancestors
  If ordering fails → invalidate lineage segment.

6.5 Storage Requirements

6.5.1 Off-Chain Storage

Event objects must be retained in:

- IPFS
- Filecoin
- enterprise storage
- encrypted local archives
  Systems must provide:
- content-addressability
- retrieval guarantees
- integrity preservation

6.5.2 On-Chain Storage

Only commitment proofs are stored on Kaspa.
No full events, inputs, outputs, or weights are ever placed on-chain.

6.6 Serialization and Canonical Encoding

Event objects, proofs, and anchoring payloads must use **canonical serialization**, meaning:

- deterministic field order
- consistent encoding
- preserved null fields
- no extraneous whitespace
- UTF-8 string encoding
  The recommended format is canonical JSON, but CBOR or Protobuf are acceptable in implementations.

6.7 Security Boundaries

Formal boundaries of trust in PoCI:

- **The model's private key must remain uncompromised.**
- **Kaspa's PoW DAG must remain secure and decentralized.**
- **Off-chain event data must be retrievable and unaltered.**
- **Verifiers must adhere strictly to the verification rules.**
  The adaptive controller (Section 5) enforces ongoing integrity over time but does not replace the cryptographic guarantees defined here.

7. Kaspa Integration (Consensus Layer Matching)

PoCI is chain-agnostic in principle, but its design aligns uniquely with the architectural properties of the **Kaspa Proof-of-Work blockDAG**.
Kaspa provides the throughput, ordering guarantees, timestamp integrity, and decentralization required for high-frequency, low-latency compute anchoring. No other blockchain architecture currently offers this combination of properties.

This section details how PoCI integrates with Kaspa, why Kaspa's blockDAG is the optimal choice, and how anchoring, RPC interaction, and finality contribute to system security.

7.1 Why Kaspa? Architectural Fit

PoCI requires an underlying layer that provides:

1. **Predictable, physics-anchored timestamps**

2. **High anchoring throughput with minimal congestion**
3. **Secure and immutable ordering**
4. **Fair launch and censorship resistance**
5. **Efficient fee market for micro-commitments**
   Kaspa satisfies all five requirements through:

7.1.1 BlockDAG Parallelism

Unlike linear chains, Kaspa accepts multiple blocks per second, enabling:

- rapid anchoring
- event-dense workloads
- parallel commitments from many models
  This prevents bottlenecks common to PoW and PoS chains.

7.1.2 Merge-Depth Finality

Kaspa's finality rules ensure that after a small number of DAG layers:

- ordering cannot be rewritten
- timestamps cannot be backdated
- commitments are irreversible
  This property is essential for compute-integrity systems.

7.1.3 Pure Proof-of-Work Security

PoS systems allow timestamp manipulation, governance capture, and re-org incentives.
Kaspa's PoW construction ensures:

- no privileged validator class
- no predictable manipulation of time
- no staking-centered economic attacks
- no delegated authority over model integrity
  AI provenance demands this neutrality.

7.1.4 Sub-10 Second Block Times

Low-latency PoW anchoring allows near–real-time compute provenance.

7.1.5 Simple, Efficient Transaction Model

Kaspa's UTXO model and flexible data-embedding enable compact PoCI commitments.

7.2 Anchoring PoCI Proofs on Kaspa

PoCI commits each compute event via a lightweight anchoring transaction.
This transaction contains:
commit_hash
lineage_hash
model_id
signature
Kaspa handles:

- transaction propagation
- block inclusion
- DAG ordering
- timestamp assignment
- finality enforcement
  PoCI does not modify Kaspa's consensus rules - it uses them.

7.3 Kaspa RPC Integration

PoCI interacts with Kaspa through standard RPC calls:

7.3.1 Submitting Anchors

- `submittransaction`
  Used to broadcast PoCI commitments.

7.3.2 Retrieving Anchored Proofs

- `gettransaction`
- `getblockdaginfo`
- `getblock`
- `getrawtransaction`
  These provide ancestry, ordering, and timestamp metadata for verification (Loop 4).

7.3.3 Verifying Finality

Kaspa's merge-depth rules allow:

- checking whether an event is final
- rejecting suspicious out-of-order anchors
- correlating local lineage with DAG ordering
  RPC is the primary bridge between PoCI and the network.

7.4 Throughput & Scalability Analysis

PoCI anchoring requires extremely lightweight transactions.
Assuming typical PoCI proof payloads of **110–170 bytes**, Kaspa’s blockDAG provides:

- thousands of PoCI anchors per second globally
- support for hundreds of high-frequency AI clusters
- negligible fee impact
- minimal chain bloat
  Kaspa’s high throughput and low-latency confirmation make it suitable for compute pipelines producing many commitments per minute.

7.5 Timestamp Integrity

Kaspa’s PoW-anchored timestamps provide guarantees critical to PoCI:

7.5.1 Non-Forgeability

A model cannot anchor an event “in the past:”

- timestamp must satisfy DAG rules
- anchor cannot be inserted before older blocks
- merge-depth prevents retroactive reordering

7.5.2 Consensus-Validated Time

Block timestamps reflect:

- network median time rules
- miner competition
- propagation physics
  They cannot be influenced by a single actor.

7.5.3 Drift Detection

By comparing:

- local timestamp
- on-chain timestamp
  PoCI can detect manipulation attempts (Loop 4 + AIC).

7.6 DAG Ordering as a Lineage Integrity Primitive

Kaspa’s blockDAG provides *partial ordering*, which gives PoCI a natural fit:

- multiple commitments at the same moment

- consistent parent/child ordering
- no ambiguity about relative event sequencing
- impossible retroactive insertion of “older events”
  The DAG acts as a global truth graph mirroring the model’s local lineage chain.
  This alignment makes the system mathematically elegant:
  the model builds a *local hash-chain*, and the DAG provides a *global hash-ordering*.

7.7 Fee Considerations

PoCI relies on frequent micro-commits.
Kaspa’s fee environment is ideal because:

- fees are negligible for lightweight data
- block throughput prevents fee congestion
- anchoring thousands of events daily costs pennies
- cost does not scale destructively with adoption
  This makes PoCI economically viable at scale.

7.8 Future Extensions with Kaspa VPs (Virtual Programs)

Kaspa’s planned Virtual Programs may enable:

- programmable validation of PoCI commitments
- on-chain logic for auditing rules
- selective model attestations
- event consistency constraints
- partial zero-knowledge verification on-layer
  PoCI is designed to be compatible with VP-based extensions without requiring changes to the underlying protocol.

7.9 Philosophical Alignment

PoCI’s goals mirror Kaspa’s philosophy:

- decentralization
- neutrality
- permissionlessness
- timestamp truth
- transparency
- censorship resistance
- real-world utility
  Kaspa provides the physics-anchored substrate PoCI needs to ground computational truth

in immutable reality.

7.10 Summary

Kaspa is not just a suitable base-layer - it is the *optimal* foundation for PoCI.
Its BlockDAG structure, fast finality, PoW timestamp integrity, and high throughput make it uniquely capable of supporting a scalable, trustless AI compute-integrity framework.
PoCI is intentionally designed to leverage Kaspa's architecture without modifying the protocol, ensuring compatibility, forward extensibility, and decentralized security.

8. Security Model

PoCI is designed for adversarial environments where models, operators, and external actors may attempt to manipulate computational history, falsify events, forge proofs, or strategically hide behavior.
The security model defines PoCI's assumptions, adversary capabilities, and the guarantees PoCI provides under those assumptions.
PoCI's defense-in-depth arises from the layered design of:

- the Four-Loop compute-integrity pipeline,
- Kaspa's PoW-based timestamp and ordering guarantees, and
- the Adaptive Integrity Controller (AIC), which continuously monitors for behavioral anomalies.

8.1 Trust Assumptions

PoCI relies on the following fundamental assumptions:

8.1.1 Cryptographic Assumptions

- Ed25519 signatures remain unforgeable.
- SHA-256/BLAKE3 hash functions remain collision-resistant.
- Model private keys are not compromised.

8.1.2 Network Assumptions

- Kaspa's PoW consensus remains secure and decentralized.
- No single actor can control a majority of network hashpower.
- DAG ordering rules remain resistant to manipulation.

8.1.3 Storage Assumptions

- Off-chain event data is retained by the model provider or an external auditor.

- Off-chain data matches the commit_hash when rehashed.

8.1.4 Verification Assumptions

- Verifiers follow the protocol's verification rules.
- The AIC is correctly implemented by auditors and monitoring agents.
  PoCI does **not** require trusting the model operator - only the cryptographic and network foundations.

8.2 Adversarial Goals

PoCI considers adversaries who may attempt to:

1. **Forge events**
2. **Backdate computations**
3. **Suppress or omit events**
4. **Insert fabricated events into lineage**
5. **Rewrite model history**
6. **Manipulate timestamp differences**
7. **Compromise model identity keys**
8. **Emit deceptive or misleading metadata**
9. **Utilize replay attacks**
10. **Exploit predictable verification rules** (AIC mitigates this)
    The design is intentionally adversarial-first. Trust is to be gained; through lineage-and proofs.

8.3 Threat Categories

8.3.1 Local Forgery Attacks

Goal: Create fake events not actually produced by the model.
Mitigated by:

- commit_hash integrity
- Ed25519 signatures
- lineage hashes
- Loop 4 verification rules
  Attack feasibility: **Low**

8.3.2 Backdating Attacks

Goal: Create the illusion a computation occurred earlier than it did.
Mitigated by:

- Kaspa's immutable PoW timestamps
- DAG ordering
- merge-depth finality
- timestamp drift checks
- AIC pursuit of anomalies
  Attack feasibility: **Extremely low** once anchored.

8.3.3 Event Omission (Suppression) Attacks

Goal: Hide specific compute events to obscure undesirable actions.
Mitigated by:

- lineage-pointer continuity
- missing-event detection
- AIC behavioral drift signals
- forced anchoring rules
- divergence between expected and observed commit frequency
  Attack feasibility: **Medium**, but detectable.

PoCI cannot force honest event creation - but it *can detect when events are missing or when recorded behavior does not match expected compute patterns.

8.3.4 Lineage Tampering

Goal: Modify or reorder event history.
Mitigated by:

- lineage_hash
- prev_commit_hash pointers
- DAG ordering consistency checks
- Loop 4 canonical reconstruction
- Kaspa immutability guarantees
  Attack feasibility: **Near zero** after anchoring.

8.3.5 Timestamp Drift Manipulation

Goal: Modify local timestamps to reduce suspicion or falsify temporal sequence.
Mitigated by:

- cross-validation of local vs. on-chain timestamps
- drift magnitude Dt flowing into 12g-AFL
- increasing verification pressure
- behavioral anomaly scoring
  Attack feasibility: **Detectable** with AIC.

8.3.6 Replay Attacks

Goal: Resubmit older proofs to create misleading lineage.
Mitigated by:

- lineage_hash uniqueness
- sequential prev_commit_hash pointers
- mismatch with on-chain DAG ordering
- Loop 4 rejection of repeated events
  Attack feasibility: **Low**

8.3.7 Key Compromise

Goal: Adversary steals a model’s private key.
Mitigated by:

- rotating model identity keys
- secure key-handling requirements
- AIC anomaly detection for abrupt behavioral change
- verifiers requiring attestation of key provenance (optional)
  Attack feasibility: **Dependent on operator security practices**
  -but detectable once behavior shifts.

8.3.8 Off-Chain Storage Manipulation

Goal: Modify or replace event data stored off-chain.
Mitigated by:

- commit_hash mismatch
- signature mismatch
- lineage rehash failure
- Loop 4 integrity verification
  Attack feasibility: **Extremely low**

8.4 Security Provided by Kaspa Integration

Kaspa’s blockDAG introduces three critical guarantees:

8.4.1 Immutability Against Rewrites

Merge-depth finality ensures events older than the finality window cannot be altered without a majority attack.

8.4.2 Order Integrity

Adversary cannot reorder commitments after anchoring, blocking fabricated earlier events.

8.4.3 Non-Manipulable Time

PoW-based timestamps are constrained by consensus rules and network time.
Kaspa provides the **physics-anchored truth surface** PoCI relies on.

8.5 Role of the Adaptive Integrity Controller in Security

The AIC enhances security by:

- learning model behavior
- detecting long-term drift
- flagging subtle anomalies
- raising verification strictness dynamically
- estimating forgery likelihood via 12g-AFL
- predicting required scrutiny via 12g-ACE
  Static systems fail against adaptive adversaries.
  AIC prevents adversaries from finding stable exploitation strategies.

8.6 Security Boundaries (What PoCI Does NOT Do)

PoCI does **not** attempt to:

8.6.1 Verify model correctness

It ensures *integrity of computation*, not correctness of reasoning.

8.6.2 Reveal internal model weights

Weights remain private; only their hash may be committed.

8.6.3 Verify training data provenance

Only compute events are tracked.

8.6.4 Prevent intentional misclassification

Model dishonesty is detectable, not preventable at the logic level.

8.6.5 Protect private keys

PoCI assumes private keys are handled securely.

8.6.6 Replace ZK-proof systems

PoCI can integrate ZKML, but is not itself a correctness oracle.
This section clarifies the protocol's scope and avoids overclaiming.

8.7 Summary

PoCI provides strong guarantees against:

- falsification
- backdating
- lineage rewriting
- timestamp manipulation
- replay attacks
- subtle behavioral drift
- long-term adversarial strategy
  By combining:
- model-side signatures,
- tamper-resistant hashing,
- DAG-based ordering,
- PoW timestamp integrity, and
- a dynamic adaptive integrity controller,
  PoCI establishes a trustless, cryptographically anchored, adversarial robust system for AI compute provenance.

9. Limitations

Every integrity system makes design decisions that introduce tradeoffs. PoCI aims to be a practical, scalable, privacy-preserving framework for AI compute integrity, but it necessarily inherits limitations from its architectural choices and underlying assumptions.
This section details those limitations and the contexts in which PoCI may require supplementation or alternative mechanisms.

9.1 PoCI Does Not Prove Correctness of Computation

PoCI verifies that a computation **happened**, *when* it happened, and *who* produced it - but it does **not** verify:

- that the computation was *correct*,
- that the model behaved ethically or safely,
- that the model used appropriate data,
- or that internal reasoning followed expected logic.
  PoCI ensures **integrity**, not **validity**.

For correctness guarantees, PoCI must be paired with:

- formal verification,
- zero-knowledge inference proofs,
- behavioral audits,
- or trusted model evaluation frameworks.

9.2 Reliance on Off-Chain Storage

PoCI intentionally minimizes on-chain footprint.
This requires off-chain storage to:

- retain full event objects,
- make them available for auditors,
- maintain access durability.
  If off-chain data is lost, the commit*hash will still prove the event _existed*, but full semantic reconstruction may be impossible.
  This is a design choice favoring scalability over total on-chain self-containment.

9.3 Local Event Creation Cannot Be Forced

PoCI assumes that models emit events truthfully.
This is a known limitation in all provenance systems: **you cannot force an adversary to record an event they wish to hide.**
PoCI's mitigations include:

- anomaly detection
- timing irregularities
- missing lineage links
- expected compute-frequency baselines
- AIC suspicion scoring
- auditor-side consistency checks
  But if a model or operator chooses not to record an event, PoCI can **detect the pattern**, not reconstruct the missing computation.

9.4 Key Management Remains an External Risk

If a model's private key is compromised:

- attackers can forge events,
- impersonate the model,
- or create false lineage.
  PoCI provides **detection**, not **prevention**, of such compromise.
  Operator key management practices and optional hardware attestation become essential.

9.5 Timestamp Drift Tolerance Has Practical Limits

PoCI's timestamp cross-validation is extremely powerful, but:

- network delays,
- batching behavior,
- queueing,
- and asynchronous models
  can all produce legitimate timestamp drift.
  Therefore, thresholds must be:
- generous enough to avoid false positives,
- strict enough to detect manipulation.
  This introduces a tunable tradeoff between **security** and **usability**.

9.6 DAG Ordering is Strong but Not Omnipotent

Kaspa's DAG provides:

- partial ordering
- immutable finality
- non-manipulable timestamps
  But it does not enforce:
- total order across all global commitments,
- auditor-specific sequencing rules,
- or infer meaning behind ordering patterns.
  This is by design, but auditors must interpret ordering with awareness of the DAG's partial structure.

9.7 Adaptive Controller Is Not a Silver Bullet

The AIC can detect behavioral drift, but:

- it can be tuned poorly,
- it can be bypassed by models that learn slow-evasion tactics,

- it depends on high-quality anomaly scoring,
- it may generate false positives during benign workload shifts.
  AIC raises the cost of misbehavior - it does not eliminate the possibility.

9.8 Dependence on Kaspa's Network Health

PoCI's guarantees depend on:

- Kaspa's hash rate,
- decentralization,
- miner distribution,
- DAG security,
- propagation latency.
  If Kaspa were to become centralized or compromised, PoCI's guarantees would degrade accordingly.
  This is no different from any system built on top of a consensus layer.

9.9 Economic Tradeoffs

While Kaspa's fees are low, extremely computedense deployments may still incur:

- cumulative transaction costs,
- burst-related fee pressure,
- chain bloat concerns (long-term),
- resource demands on verifiers.
  PoCI chooses **economic feasibility** over zero-cost purity - a conscious tradeoff.

9.10 PoCI Is a Complement, Not a Replacement

PoCI cannot replace:

- behavioral safety techniques
- training data oversight
- sandboxing
- full chain-of-thought audits
- human governance
- external scrutiny
  It is a **layer**, not a full-stack solution.
  PoCI provides *integrity primitives* - not moral, ethical, or alignment guarantees.

9.11 Summary

PoCI makes deliberate tradeoffs:

- **off-chain storage** for scalability
- **low data footprint** for chain integrity
- **partial visibility** to preserve privacy
- **dynamic verification pressure** over static models
- **timestamp anchoring** instead of correctness proofs
  These tradeoffs enable PoCI to scale across real-world AI workloads, but they introduce limitations that must be acknowledged.
  PoCI is not a universal solution - it is a practical, cryptographically grounded framework for ensuring *when*, *by whom*, and *in what order* AI computations occurred, even in adversarial conditions.

10. Future Work

PoCI provides a foundational framework for AI compute integrity, but it is intentionally designed to evolve.
As cryptography, AI architectures, and the Kaspa ecosystem advance, PoCI can integrate new techniques that expand its capabilities while maintaining the core principles of decentralization, minimal on-chain footprint, and adversarial robustness.

This section outlines promising directions for future research, protocol integration, and applied development.

10.1 Zero-Knowledge Integration (ZKML + PoCI)

PoCI currently verifies:

- authorship,
- ordering,
- lineage,
- and timing.
  But it does **not** verify correctness of the computation or the inputs themselves.
  Future versions may integrate:

10.1.1 ZK-Inference Proofs

Allowing models to produce:

- cryptographic proofs that an inference was computed correctly
- without revealing inputs, outputs, or weights

10.1.2 ZK-Model Attestation

Selective proofs that:

- a specific model version was used
- weights match the committed hash
- no unauthorized tampering occurred

10.1.3 ZK-Partial Reveals

Verifiable assertions about:

- safety checks
- redaction compliance
- classification correctness
  All without revealing actual content.
  This would elevate PoCI from *integrity* to *verifiable correctness*.

10.2 Kaspa Virtual Programs (VPs)

Kaspa’s planned Virtual Programs introduce programmable logic.
PoCI could leverage VPs to enable:

- on-chain lineage consistency checks
- scheduled or conditional audit triggers
- multi-model consensus rules
- automated timestamp-drift penalties
- registries of trusted model identities
- proof bundling or commitment batching
  This would further decentralize the verification layer.

10.3 Hardware Attestation Integration

PoCI currently assumes SK (secret key) security.
Future enhancements may incorporate:

- TPM or TEE-based identity attestation
- secure enclaves for event signing
- hardware-proved model integrity
- remote attestation for edge devices
  This is particularly important for:
- robotics
- safety-critical AI
- IoT models
- agent systems with physical effects

10.4 Federated & Multi-Agent PoCI

PoCI currently treats each model independently.
Future extensions may support:

- collaborative multi-agent workflows
- federated lineage graphs
- cross-model commitment stitching
- multi-party signatures on shared compute events
  This supports:
- autonomous fleets
- decentralized swarms
- distributed model ecosystems

10.5 DAG-Level Aggregation & Compression

Future research may enable:

- commitment aggregation
- rolling hashes for model sequences
- tree-based proofs for bulk events
- periodic on-chain checkpoints
- DAG-level compression for PoCI-heavy workloads

10.6 Behavioral Analytics & Trust Networks

Building on the Adaptive Integrity Controller, future extensions include:

10.6.1 Global Trust Index

A decentralized network of verifiers collaboratively maintaining:

- trust scores
- anomaly classifications
- model reputations

10.6.2 Behavioral Fingerprinting

Statistical identity models for:

- output patterns
- timing signatures

- characteristic event frequency
  Useful in detecting subtle evasion strategies.
  This preserves scalability as adoption increases.

10.7 Dataset Provenance Extensions

PoCI tracks compute provenance - not data provenance.
A potential extension is:

- dataset hashing
- versioning
- lineage of training data
- incremental dataset mutation proofs
  This would expand integrity coverage into model development.

10.8 Integration with L2 and Off-Chain Compute Networks

PoCI can be extended to integrate with:

- Kaspa-aligned rollups
- off-chain compute clusters
- L2 scaling networks
- MPC or federated learning clusters
  Each compute shard could anchor via PoCI, forming global lineage.

10.9 Formal Verification of PoCI Protocol

Longer-term work includes:

- modeling PoCI as a formal system
- proving invariants around lineage integrity
- analyzing adversarial edge cases
- proving bounds on forgery-detection latency
- verifying correctness of 12g-ACE and 12g-AFL under assumed adversary models
  This elevates PoCI from design to mathematically provable system.

10.10 Community & Ecosystem Growth

PoCI is positioned to become a building block of:

- AI governance frameworks
- model-trust infrastructure
- decentralized attestation networks

- industry standards for audit trails
  Future work includes:
- open-source reference implementations
- standard event schemas
- RPC & client libraries
- developer tooling
- academic collaborations
- integrations with AI platforms

10.11 Summary

PoCI is intentionally modular, extensible, and forward-compatible.
Future work includes deeper integration with cryptographic proofs, programmable on-chain logic, multi-agent lineage, formal verification, and global trust networks.
PoCI’s long-term vision is to establish a universal framework for **verifiable AI behavior**, grounded in cryptographic integrity and decentralized timestamp truth.

11. Conclusion

Modern AI systems operate at a scale, speed, and complexity that make traditional trust models obsolete.
Models can fabricate results, obscure lineage, misrepresent when computations occurred, and evolve faster than oversight mechanisms can adapt. Existing provenance systems rely on centralized infrastructure or mutable logs, providing no cryptographic proof that computations were performed honestly or in the claimed sequence.

Proof of Compute Integrity (PoCI) addresses this foundational gap by introducing a decentralized, cryptographically grounded framework for verifying *when*, *by whom*, and *in what order* AI computations occurred. Through its Four-Loop architecture - event creation, proof generation, on-chain anchoring, and verification - PoCI transforms opaque model behavior into verifiable, tamper-resistant evidence.

By anchoring compute commitments to the **Kaspa Proof-of-Work blockDAG**, PoCI derives strong guarantees of timestamp integrity, immutable ordering, and decentralized trust. This makes it possible to construct a complete, cross-verifiable compute lineage graph without exposing sensitive model parameters or data.

Crucially, PoCI moves beyond static audit systems through the **Adaptive Integrity Controller (AIC)**. The AIC incorporates PoCI’s original formulations - **12g-ACE (Adaptive Compute Expectation)** and **12g-AFL (Adaptive Forgery Likelihood)** - to create a dynamic trust-pressure system. This adaptive layer detects behavioral drift, increases scrutiny when anomalies occur, and aligns PoCI with learnability-equilibrium principles rather than static game-theoretic assumptions.

PoCI is not designed to prove model correctness or ethical alignment, nor does it enforce visibility into proprietary data or logic. Instead, it establishes the **integrity substrate** required for higher-level AI governance, auditing, and interoperability. It is intentionally modular, privacy-preserving, and scalable, allowing future integration with zero-knowledge inference, hardware attestation, dataset lineage systems, and Kaspa Virtual Programs.

As AI systems continue to grow in power and autonomy, society requires a reliable means of verifying computational truth - a method to ensure that models operate with cryptographic accountability rather than unverifiable self-reporting.
PoCI represents a step toward that future: a principled, decentralized framework that anchors intelligence to reality through immutable proof.

By bridging modern AI behavior with physics-based consensus, PoCI lays groundwork for a new paradigm in AI trust - one where integrity is not assumed, but verifiably proven.

Appendix A — Notation & Symbols

This appendix defines the notation used throughout PoCI, including all mathematical symbols, cryptographic values, and controller variables.

Symbol	Meaning
H(·)	Cryptographic hash function (SHA-256 or BLAKE3)
skmodel	Model private key
pkmodel	Model public key
commit_hash	Hash of serialized event
lineage_hash	Hash of commit + previous commit
prev_commit_hashprev	Parent event hash
Tlocal	Local timestamp of event creation
Tchain	Kaspa block timestamp
Dt	Timestamp drift
At	Anomaly score at time t
Ct	Consistency score
Ft	Forgery likelihood
Pt	Verification pressure
α,β,γ,δ,ϵ	Sensitivity coefficients in 12g equations

Appendix B — Canonical JSON Encoding Rules

All events must be serialized via *canonical JSON*.
Requirements:

1. Lexicographically sorted keys
2. No whitespace beyond structural minimum
3. Null fields explicitly encoded as `null`
4. UTF-8 encoded
5. Deterministic number formatting (no scientific notation)
   This ensures identical hashing across implementations.

Example:

```
{
"event_id": "42",
"input_class": "vision:object",
"metadata": {
"task": "detection",
"version": "1.3.1"
},
"model_id": "kaspa:pkh:0xabc123...",
"output_hash": "0x7fa...d9e",
"prev_event_id": "41",
"timestamp_local": 1731579874,
"weights_hash": null
}
```

Appendix C — Reference Implementation Pseudocode

C.1 Event Creation (Loop 1)

```
def create_event(model_id, prev_id, weights_hash, input_class, output_hash, metadata):
event = {
"model_id": model_id,
"event_id": new_nonce(),
"prev_event_id": prev_id,
"timestamp_local": now(),
"weights_hash": weights_hash,
"input_class": input_class,
"output_hash": output_hash,
"metadata": metadata
}
return canonical_serialize(event)
```

C.2 Proof Generation (Loop 2)

```
def generate_proof(event, sk_model, prev_commit_hash):
commit_hash = H(event)
```

```
lineage_hash = H(commit_hash + prev_commit_hash)
signature = sign(sk_model, commit_hash)
```

```
return {
    "commit_hash": commit_hash,
    "lineage_hash": lineage_hash,
    "model_id": extract_model_id(event),
    "signature": signature,
    "prev_commit_hash": prev_commit_hash
}
```

C.3 Verification (Loop 4)

```
def verify_event(event, proof, pk_model):
if H(event) != proof["commit_hash"]:
return False
if not verify(pk_model, proof["commit_hash"], proof["signature"]):
return False
if H(proof["commit_hash"] + proof["prev_commit_hash"]) != proof["lineage_hash"]:
return False
return True
```

Appendix D — Adaptive Controller Example Profiles

This appendix contains *baseline anomaly scoring examples* for implementers building 12g-ACE / 12g-AFL systems.

Category	Description	Weight
Missing event	unexpected gap in lineage	3
Timestamp drift	deviation beyond tolerance	2
Replayed commit	identical commit_hash	4
Late anchoring	large delay T_chain – T_local	1
Metadata inconsistency	task-type mismatch	1
Signature anomaly	failure or key rotation jumps	5

D.2 Example Consistency Signals

Signal	Meaning	Score
Stable event frequency	expected workload	+3
Minimal drift	< threshold	+2

Signal	Meaning	Score
Unbroken lineage chain	no gaps	+3

These feed into:

- 12g-ACE for verification pressure
- 12g-AFL for forgery likelihood

Appendix E - Example Threat Analysis Cases

This appendix contains concrete adversarial scenarios to help implementers understand edge cases.

E.1 Omitted-Event Scenario

- Event 53 is missing
- Lineage jumps from 52 to 54
- AIC flags anomaly
- ACE increases pressure
- AFL increases forgery likelihood
- Auditor reconstructs expected workload → confirms omission

E.2 Backdating Attempt

- Local timestamp claims T0
- Kaspa anchor timestamp is much later
- drift D > threshold
- AFL spikes immediately
- Model enters high-scrutiny mode

E.3 Replay Attack

- commit_hash reused for a new anchoring
- prev_commit_hash mismatch
- PoCI rejects instantly
  These examples reinforce PoCI’s adversarial strength.

Appendix F — Extended Event Schema Variants

For advanced use cases, optional fields include:

- `model_version`

- `agent_cluster_id`
- `model_role` (e.g., planner, critic, summarizer)
- `compute_cost_estimate`
- `safety_flag`
- `zk_proof_ref`
  Fields remain off-chain; only the hash is anchored.

Appendix G - Kaspa Anchoring Payload Examples

Minimal Payload Example

0x01 | commit_hash | lineage_hash | signature | model_id

Verbose Payload for Future VPs

```
{
"commit_hash": "...",
"lineage_hash": "...",
"signature": "...",
"model_id": "...",
"anchor_version": 1,
"vp_flags": [ "lineage-check", "timestamp-check" ]
}
```

Appendix H - Glossary of Key Concepts

A quick-reference glossary for new readers:

- **PoCI** - Proof of Compute Integrity
- **AIC** - Adaptive Integrity Controller
- **Compute Event** - a single AI computation record
- **Commit Hash** - cryptographic fingerprint of the event
- **Lineage** - the sequence of events linked over time
- **DAG** - Directed Acyclic Graph (Kaspa’s block structure)
- **Merge Depth** - finality rule enforcing ordering stability
- **12g-ACE** - Adaptive Compute Expectation equation
- **12g-AFL** - Adaptive Forgery Likelihood equation
- **VP** - Virtual Program (future Kaspa extension)

Appendix I — Credits & Original Contributions

Original Contributions Introduced in This Work

1. **Four-Loop Compute Integrity Architecture**
2. **Adaptive Integrity Controller (AIC)**
3. **12g-ACE — Adaptive Compute Expectation Equation**
4. **12g-AFL — Adaptive Forgery Likelihood Equation**
5. **Event-lineage hashing method combining local and chain anchoring**
6. **Decentralized AI lineage reconstruction over Kaspa blockDAG**
7. **Learnability-equilibrium alignment for integrity systems**

References

[1] Satoshi Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.

[2] Yonatan Sompolinsky, Aviva Zohar, “Secure High-Rate Transaction Processing in Bitcoin,” GHOST Protocol, 2013.

[3] Yonatan Sompolinsky, Tal Moran, “Spectre: A Fast and Scalable Cryptocurrency Protocol,” 2016.

[4] Yonatan Sompolinsky et al., “PHANTOM: A Scalable BlockDAG Protocol,” 2018.

[5] Yonatan Sompolinsky et al., “DAGKNIGHT: A High-Security BlockDAG Protocol,” Kaspa Research Paper, 2021.

[6] Kaspa Team, “Kaspa: BlockDAG, GHOSTDAG, and the Kaspa Consensus,” kaspa.org/tech.

[7] Dwork & Naor, “Pricing via Processing or Combatting Junk Mail,” Proof-of-Work origin, 1992.

[8] Daniel J. Bernstein et al., “Ed25519: High-Speed High-Security Signatures,” RFC 8032.

[9] NIST, “SHA-256 Secure Hash Algorithm,” FIPS PUB 180-4.

[10] Canonical JSON Specification, “IETF Draft: Deterministic JSON Encoding,” 2020.

[11] ZKML Research Group, “Zero-Knowledge Machine Learning,” zkml.org (overview).

[12] G. Bentov et al., “Blockchain Timestamping: Attacks and Countermeasures,” 2016.

[13] Carlini et al., “Adversarial Machine Learning: Attacks and Defenses,” Google Brain, 2019.

[14] Ribeiro et al., “Anomaly Detection in Machine Learning Systems,” 2020.

[15] Ronen et al., “Machine Learning Model Provenance and Accountability,” 2021.

[16] OpenAI, Anthropic, Google DeepMind — Safety & Provenance Reports (general provenance landscape).

[17] Kaspa Team, “BlockDAG Architecture & High-Throughput Design,” Technical Notes, 2023.

[18] 12gauge & ChatGPT, “Proof-of-Compute-Integrity (PoCI) Architecture & Adaptive Controller,”
This Work, 2025.

*This work was conceived and architected by 12GaugeKenshin. GPT-based reasoning and drafting
tools were used to assist in organizing concepts, refining language, and formalizing the mathematical and structural components of the system. All core ideas, system design choices, and final definitions originate from the primary author.